



REAL 3D



Created in REAL 3D V2.4 by Grant Neisner, reflex llc

USER MANUAL ***Version 3***



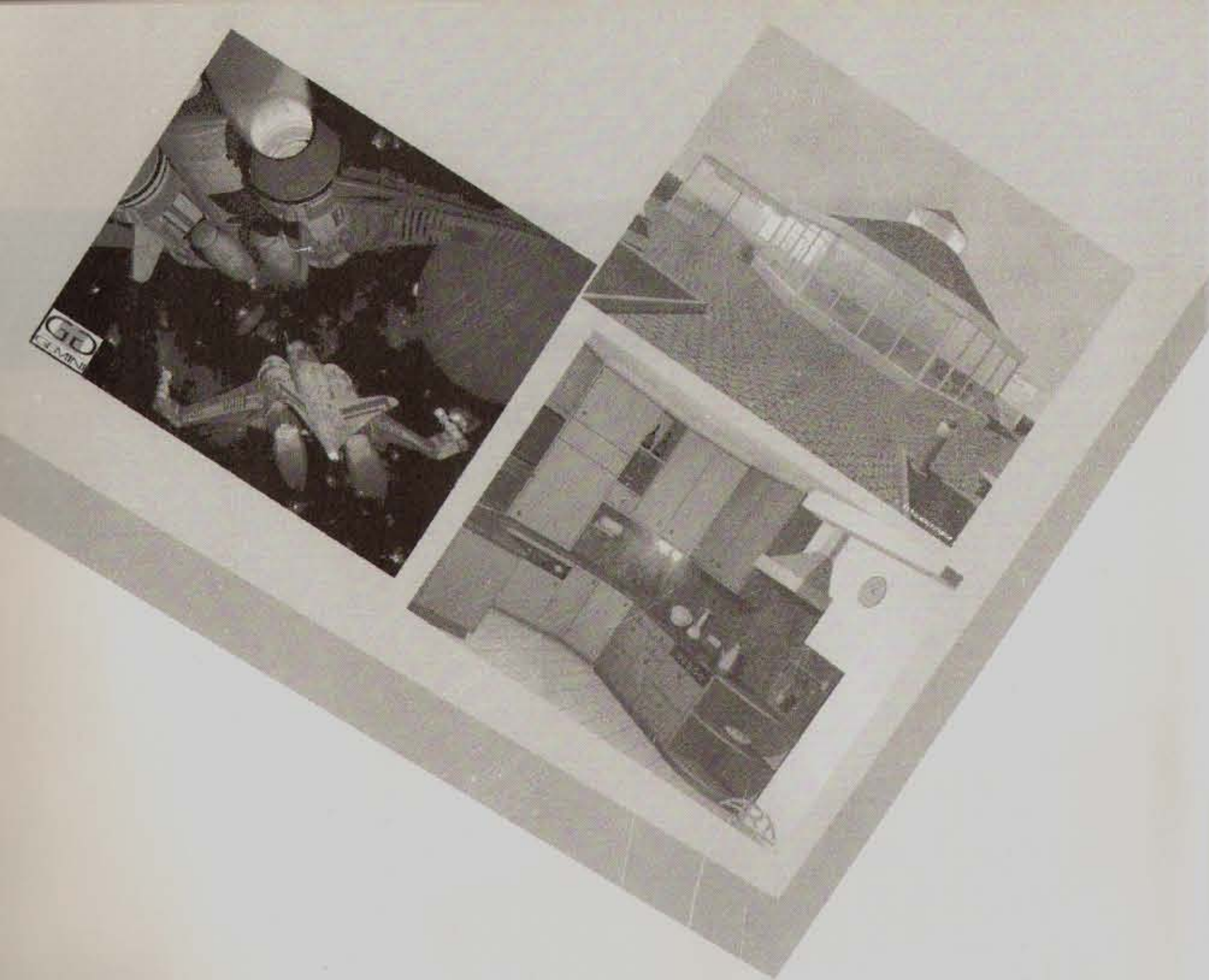
REAL 3D

version 3

User Manual



Realsoft



REAL 3D User Manual for version 3.
Copyright© 1995 Realsoft Oy. All rights reserved.
REAL 3D is a trademark of Realsoft Oy. All other brand and product names are trademarks of their respective companies.
Printed in Finland by WSOY.
First edition.

CONTENTS

1. TUTORIAL

Getting started with Real 3D v.3	7
Modelling	10
Rounding	10
Surface coordinates	11
Freeform modelling	12
Fractals	15
Materials	16
Glow example: a space ship	16
One way mirror	17
Animation	18
Path animations	18
A simple keyframe animation	20
Creating hierarhical animations using keyframing	20
Pivot points	22
Animated fade-out	23
Skeletons	24
Fidelity	26
Hierarchical skeletons	27
Morphing skeletons	29
Surface method	30
Shrink wrapping	32

2. MENU FUNCTIONS

File/	33
Objects/Load	33
Project/Insert Sections, Save Sections, Replace Sections	33
Materials/Window	33
Edge X/Y	33
Freq X and Y	33
Texture antialiasing	34
Glow	34
Depth scope handler	34
Angle scope handler	34
Roughness bump handler	34
Dither color handler	35
Material color	35
Relative texture coordinates	35
New image formats	36

Materials/Purge	36
Windows/ViewTool	36
Create/	38
Structure/Link	38
Structure/Group	38
Structure/Method	39
Controls/Helix	40
Controls/Skeleton	40
Friction	41
Fidelity	41
Angle Constraints	41
Sub skeletons	42
Fractals/Landscape	43
Fractals/Tree	43
Modify/	43
Properties/Attributes	43
Properties/Animation	43
Properties/Skeleton Attr.	44
Properties/Geometry	44
Properties/Fade	46
Special/Reflect	46
Freeform/Type	46
Freeform/Round	47
Freeform/Convert to Triset	47
Freeform/Surface to Groups	47
Freeform/Fidelity	47
View/	48
Type/Surface	48
Camera/Camera View	48
Render/Settings	48
Auto Box Rendering	48
Interlace fields	48
Super sampling	49
Alpha channel output in IFF file rendering	49
JPEG support	49
Saving and loading render settings	49
Post effects	50
Glow	51
Distance anti-aliasing	51
Render/Export DXF	51
Drawing Settings	51
Animate/	52
Create/Keyframe	52
Create/Inverse Kinematic	52
Create/Skeleton	52

Create/Shrink wrapping	53
Edit	53
Extras/	53
Vectors/Deselect Group	53
Repeat Last	54
Show selected objects -hotkey	54
Settings/	54
Attributes	54
General	54

3. ANIMATION SYSTEM

The key editor	55
Envelopes	57
Path animations	59
Keyframe method	59
Morphing method	60
Skeleton method	63
Simple skeleton method	63
Inverse kinematics method	63
Surface method	64
Shrink wrapping method	64
Transform method	66

4. RPL

Local Variables	67
Parameters	67
New RPL words	68
CS_TRANSFABS	69
Changes to RPL specification	81
A complete GUI programming example	82

5. MISCELLANEOUS

Compatibility	87
Comparison of Amiga / Windows specific features	88
Installation	88
Naming conventions	88
Menu function differences	88

INDEX

.....	93
-------	----

1. TUTORIAL

Real 3D V3 contains many new features and improvements. This chapter presents some basic examples to demonstrate these. See the reference section for more detailed descriptions.

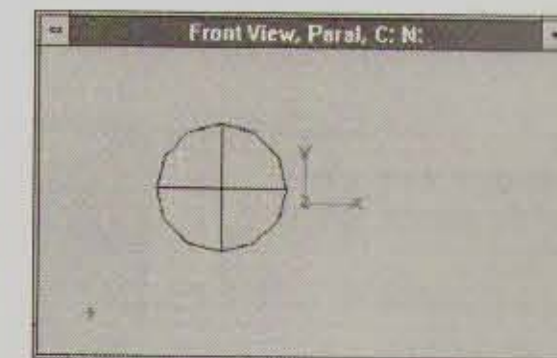
GETTING STARTED WITH REAL 3D V.3

Welcome to Real 3D v.3. This first tutorial example will take you step-by step through the process of creating an animation. If you are unfamiliar with Real 3D, the tutorials will show you how easy it is to create animations, but no attempt will be made here to explain the various functions you will be using in any detail. The reference section of the manual is the best place to look for detailed function descriptions of the actions you will perform in these tutorial examples.

So, let's create a simple animation: a moving wooden sphere.

1. Start Real 3D by clicking the 'TutorEnv' icon.
2. First we create a sphere:
 - Select Create/Visibles/Sphere from the menu.
 - Click the left mouse button once on the large 'View' window with the label 'front view'.
 - Move the mouse to see how Real 3D draws a spherical shape.
 - Click the left mouse button again to create a sphere.

Defining a sphere.



3. Next, we need to change the material of the sphere to wood.

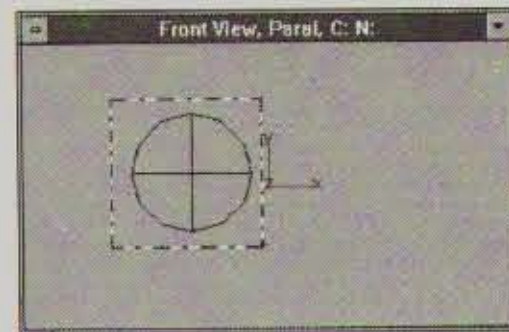
- Click on the image of the pine wood material in the material select window at the right of the screen.
- Click the 'Parallel' button at the bottom of the same window.

Now Real 3D will wait for you to draw a 'material mapping' rectangle in the view window:

- Move the mouse to a point above and to the left of the sphere and click the left mouse button.
- Move the mouse down and right until the rectangle drawn by Real 3D covers the sphere.
- Click the left mouse button again to finish the rectangle.

This will now map the wood grain pattern onto the sphere.

Defining a material mapping.



4. To make sure we have created a wooden sphere:

- Select the menu View/Render/Window.

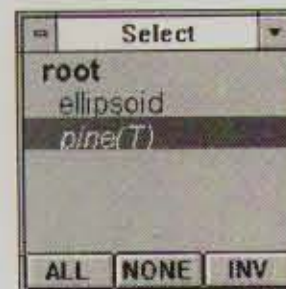
Real 3D renders a shaded image.

A wooden sphere.



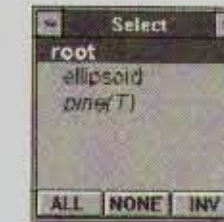
5. The 'Select' window at the top right corner of the display shows you the hierarchical structure of the wooden sphere. The 'Root' object consists of two sub objects: a sphere and a wood mapping.

The select window shows the hierarchy of the wooden sphere.



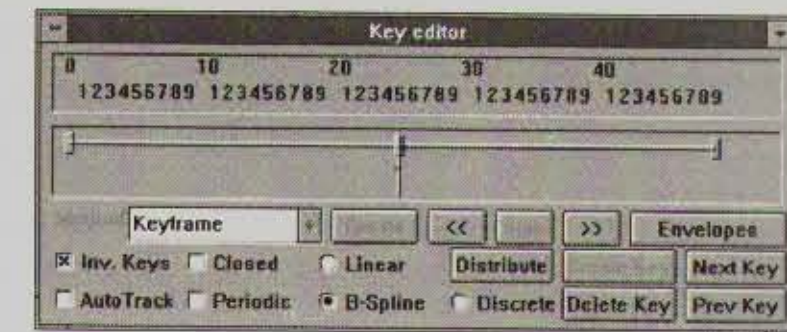
The name of the previously created 'wood' mapping object is highlighted. This means it's in a 'selected' state. We're going to animate both the sphere and the wood mapping together, therefore:

- Select the 'Root' object by clicking its name with the left mouse button on the select window.



Selecting the 'Root' object.

6. We'll use the keyframe technique to animate the sphere. Animation operations can be carried out in the key editor window.



The key editor.

- Pick 'Keyframe' from the 'Method' list of the key editor.
- Make sure the 'Root' object is still selected
- Click the 'Create' button in the key editor to start keyframing.

7. Next we will define a second keyframe.

- Move the mouse on top of the last frame index '49' displayed in the key editor and click the left mouse button.

A thin vertical line jumps to that frame. This shows the current time. Now we will create the second key frame

- Click the 'Create Key' button.

A new, small rectangle which represents the second key frame, appears at the end of the horizontal 'time line' of the key editor.

- Select the function Modify/Linear/Move from the menu.
- Click in the middle of the sphere.
- Move the object to a new position and click again.

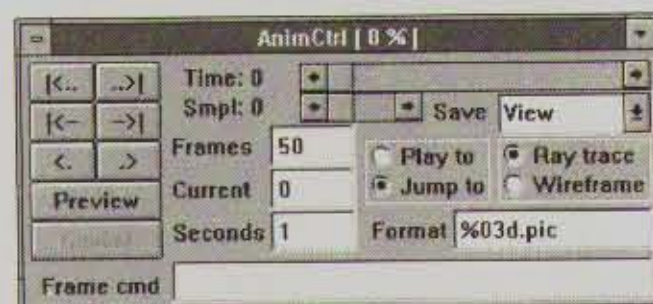
8. The second key frame was defined at the end of the animation.

- Play the animation to the beginning by clicking the '<<' button in the key editor.

The sphere moves to its original position.

9. Let's make the motion more interesting by creating a third keyframe.
 - Click on frame '25'.
 - Create a new keyframe by clicking the 'Create Key' button.
 - Move the object to a new position with the menu function Modify/Linear/Move.
10. The final step is to preview the animation.
 - Open the Animation window by selecting the menu File/Windows/Animation.
 - Make sure that the 'Ray trace' option, not 'Wire frame', is selected.
 - Set the Save field to 'View'.
 - Click the 'Preview' button.

The animation window



V3samples\
firsttut.prj

This renders and shows the animation automatically.

MODELLING

Rounding

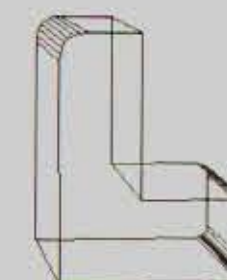


V3samples\
round.prj

The Modify/Freeform/Round function, despite its name, may be applied not only to freeform curves but also to polygon and polyhedron primitives. Let's experiment with this:

1. Select Create/Visibles/Polyhedron and draw an 'L' shape.
2. Select by shift-dragging the top left corner points of the polyhedron.
3. Select the menu Modify/Freeform/Round.
4. Enter 0.1 into the radius field.
5. Enter 6 into the point count field.
6. Click OK. The top left corner becomes rounded.
7. Shift-drag all the points at the right edge of the polyhedron.
8. Hit the '5' key (it's the hot key for 'repeat last action').
9. Enter a radius of 0.05.
10. Enter 3 as the point count for rounding and click OK. Two corners more become rounded.

A rounded polyhedron



Surface coordinates

This example demonstrates the View/Type/Surface coordinate mode, which makes it possible to draw curves directly onto object surfaces.

1. Start a new project with menu item File/Project/New.
2. Select View/Viewcam/Reset all.
3. Select View/Input Plane/Reset Hot Point.

Let's first create an object to project surface coordinates onto later. We'll use the lathe tool.

4. Select the menu Create/Compound tools/Lathe.
5. Draw a vertical axis in the middle of the view window.
6. Shape a candle stick. Use the right mouse button twice when you have finished with the lathe tool.
7. Activate the View/Type/Surface option from the menu.
8. Select Create/Controls/B-spline Knot.

It is important to define the curve with knot points instead of control points, because otherwise the typical B-spline rounding nature may dislocate the actual curve shape from the candle stick surface.



v3samples\
surfcrd.prj

9. Draw a curve in a view window.

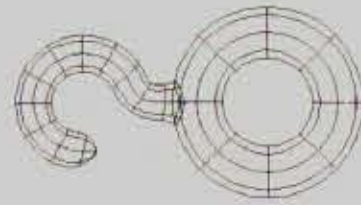
Make sure every mouse button click hits the candlestick. You can also use the cursor keys to rotate around the candlestick while you are drawing. It is best to define curve points quite densely to ensure that the created curve follows the surface as well as possible.
10. Finish the curve definition with a right mouse button click.



Freeform modelling

In this example, we'll create a freeform object consisting of a ring and a hook which is smoothly joined to the side of the ring. This example isn't intended as an introduction to B-spline modelling; read the freeform modelling section of the main manual first.

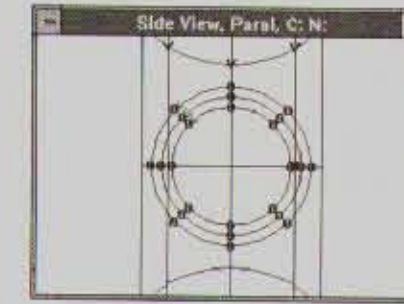
A freeform object.



1. Let's start by taking a front view:
 - Select menu View/Viewcam/Set XY.
 - Select View/Type/Parallel, which is a suitable mode for accurate modelling.
2. The ring shape is easy to create:
 - Select Create/Freeform/Torus.
 - Accept the default 8*8 point count for the mesh.
 - Shape a small circle and then a larger one.
3. Now take a suitable side view:
 - Select menu View/Viewcam/Set YZ.
 - Zoom in either by pressing '+' key several times or use the function View/Viewcam/Pos&Zoom in, until the area to which the hook will be attached is displayed in a scale which allows accurate working.
4. We'll define a smooth blend from the ring to the hook with cross-sectional building. Therefore, we need some cross section curves.
 - Select Create/Controls/B-spline Circle
 - Draw a circle in the middle of the ring.

The radius of the circle should be almost as big as the diameter of the ring (see the picture below).
5. To create the second cross-section:
 - Use Modify/Structure/Duplicate to duplicate the circle.
 - Select Modify/Linear/Size 2D.
 - Click in the middle of the circle, then at its edge.
 - Make the copied circle slightly smaller.
6. Apply the Duplicate and Size 2D functions once more. The result should be three equally centered circles as shown in the figure below.

Cross-sections for surface blending



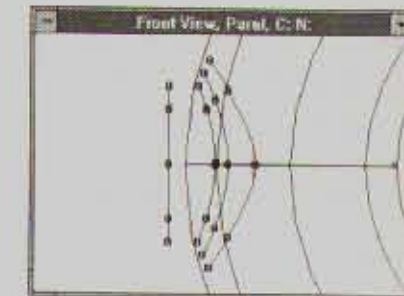
7. Duplicate the smallest circle, but don't resize it this time. Rename the copy as 'circle' to make it easily identifiable.
8. Take a front view. Move all the circles to the left side of the ring.
9. Now we will project the first three curves onto the surface of the ring with the parallel shrink wrapping function. Let's define the parameter objects carefully:
 - Select the ring mesh from the select window.
 - Press the shift key down and select the first three circles in the order they were created.
 - Release the shift key.

If you did everything correctly, there should now be four objects selected.

 - Then select menu Modify/Special/Parallel Shrink.
 - Draw a line from the middle of the circles to the right, far enough so all the points of the circles, when moved the amount defined by the line, would hit the ring.

For example, you can draw the line all the way to the middle of the ring. After the line has been defined, the circles move to the right and bend along the surface of the ring as they hit it.

Shrink wrapped circles

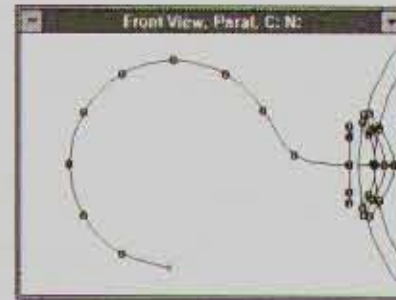


10. Parallel shrink actually projects the control points onto the surface, and the curves may now be slightly inside the ring. For the outermost curve, this is OK, because it guarantees there won't be a gap between the hook and the ring.
 - Select the two smaller projected curves
 - Apply the Modify/Freeform/Invert function to them.

This converts the control points to knot points and we get a more accurate result.

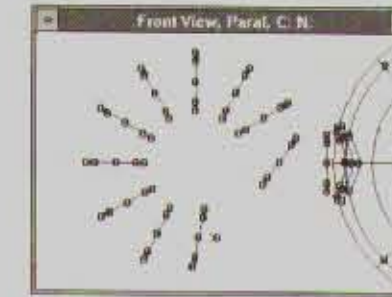
11. The actual hook will be created with orthogonal sweeping. For that purpose, we need a new curve.
 - Select Create/Controls/B-Spline knot.
 - Draw a hook-like curve starting from the middle of the unmodified circle (see the picture below).
 - Use knot point editing (Modify/Freeform/Move Knotpoint) to fine tune the shape if necessary.
 - Rename the curve as 'hook'.

A spline curve for the hook.



12. Multi-select the two curves, first 'circle' and then 'hook'. Select menu Create/Build Freeform/Orthogonal. A hook-like mesh is created.
13. Delete the 'hook' and 'circle' curves.
14. Actually, the orthogonal sweeping was used only as an intermediate step:
 - Select the hook mesh
 - Apply the function Modify/Freeform/Surface to Curves to it.
 A set of cross section curves is created.
15. Delete the hook mesh. We don't need it any more.
16. Let's analyze what kind of curves we have. First there are the three curves which were projected to the ring. After that follow cross sections from the hook mesh. Since the cross sections generated by the Surface to Curves operation represent the control polygon structure of the hook mesh, the first cross section may be partly inside the ring. If this is the case, delete it. The purpose is to get a nice ordered sequence of cross sections.
17. Continue as follows:
 - Select the last curve in the hierarchy.
 - Duplicate it twice.
 - Multi-select the last three curves, and use Modify/About COGs/Size 3D to squeeze them into a single point.
 - Move them slightly to the front of the last unsqueezed curve.
 The last three curves will generate the closing cap for the hook mesh.

Cross-sections for the hook.



18. All the curves should now be in the correct order in the hierarchy, so select them all by dragging over their names on the select window.
19. Use the menu Create/Freeform/Build from Curves. A hook mesh is recreated, this time with a smooth blending to the ring and a closing cap.
20. Delete all the curves and the object is ready.



v3samples\
hookobj.prj



Fractals

The project file v3samples\fractal.prj contains some predefined fractal trees. To use them, do the following:

1. Select File/Project/Insert sections from the menu.
2. Press the 'None' button of the opened file selector to clear all sections. Then activate the 'Fractal Trees' gadget. Click on OK.
3. Select the file 'examples/fractal.prj' using the file dialog.
4. After the file has been loaded, select Create/Fractals/Tree. This opens the fractal tree dialog.
5. Click the 'Load' button. A list dialog is opened. Select one of the predefined trees, then click OK and define the first branch for the tree by drawing first a sphere, which defines the diameter of the first branch and then a line, which defines the length and the direction of the branch. The fractal tree is then created.

6. You might also want to use a leaf object in which case create and select the object to be used as the leaf prior to activating this function.

NOTE: Delete the created tree before trying to create the next one. Otherwise you could inadvertently use the previously created tree as a leaf object for the new tree, which will lead undoubtedly to a not-enough-memory situation.

MATERIALS



v3samples/
glow.prj

Glow example: a space ship

In this example, glow feature is used to make windows of a space ship to glow and to create a smoothly glowing exhaust flame behind the ship.

1. Draw two texture maps in a paint program. The first one, 'panels' will be used as a color map to represent surface details of the ship, so it may consist of rectangular areas of grayish blue shades. The second one, 'lights', should contain suitable groups and rows of bright yellow dots (used for windows) against a black background. Save the textures to Real 3D's 'textures' drawer.
2. Model a simple space ship in Real 3D. For example, use Create/Compound tools/Lathe to create a suitable shape. The color of the ship does not matter. Rename the created object as 'ship'.
3. Use Create/Structure/Level to create a new level. Rename it as fire, and make it the current level. Create a bright orange sphere (Create/Visibles/Sphere) behind the ship, and use Modify/Linear/Extend to make it a flame like elliptic shape. The ellipsoid will be the source of the exhaust flame glow.
4. Use File/Materials/Window to open the material editor. Define a new material with the following properties:
 - Name 'panels'
 - Texture 'panels' (the texture map you just created)
 - Tile X and Y on
 - Color map and Bump map on
 - Specularity 20, Specular brightness 30-40 (to get a metal-like effect)
 Press Apply to create the material.
5. Press Reset and define another material with the following properties:
 - Name 'fire'
 - Unshaded option on (the exhaust flame doesn't require shading)
 - Glow size 90, Glow brg. 100 Press Apply.
6. Press Reset and define a third material:
 - Name 'lights'
 - Texture 'lights' (the second texture map you created)
 - Tile X and Y on

- Color map on
- Unshaded option on, to make the windows behave as if they are lights inside the spaceship
- Transp. Color option on, and the Transp color should be black R=0, G=0, B=0
- Exclusive and Scope mask on; Scope mask is important, otherwise the whole ship will glow.
- Glow size 80, Glow brg. 50

Hit Apply to create 'lights' material. Close the material editor.

7. Use Create/Mapping/Default, select 'fire' and click OK to create a mapping inside the 'fire' hierarchy level (see the hierarchy diagram below).
8. Make 'ship' the current level. Select a view so that the ship is 'end on'. Then create/Mapping/Cylinder twice and shape two cylinder maps around the ship; the first one should refer to 'panels' and the second one to 'lights'. Now the ship hierarchy should be as follows:

```

root
  ship
    part1
    part2
    ...
    panels(T)
    lights(T)
  fire
    ellipsoid
    fire(T)
  
```

9. Take a suitable view to the ship scene. Then open render settings. Select Environment or Lampless mode (you can also use Shadowless or Normal mode if you've added some light sources). Set background color to black.
10. Press POST EFFECTS button. This opens the post effect editor. Select 'Glow' from the Post effect list at the left. Then press 'CREATE' button, so that a new effect called 'Glow' is inserted to the effect list. Press EXIT to close the dialog. Then press OK to exit the render settings dialog.
11. Render the scene.

Using similar techniques to those described above allow you to add glows to either an entire object or just sections of it.

One way mirror



v3samples/
oneway.prj

Here is an interesting use for the new angle scope handler.

Just load the ONEWAY.PRJ and render to see a transparent rectangle, then use the mouse in the view window (Viewcam Pos.) to swing around to the other side. Render and the rectangle is a mirror.

This opens up some interesting possibilities for animations or tricky lighting effects.

The effect is created like this:

1. Create a rectangle in parallel XY view
2. Create the following materials with only these settings:

mirror		transp	
"mirror" (name)		"transp" (name)	
0 (specularity)		0 (specularity)	
25 (specbright)		25 (specbright)	
100 (brilliancy)		100 (brilliancy)	
0 (transparency)		100 (transparency)	
0 (turbidity)		0 (turbidity)	
0 (refraction)		100 (refraction)	
0 (currindex)		0 (currindex)	
100 (effectiveness)		100 (effectiveness)	
ANGLE scope handler		ANGLE scope handler	
a=-1.1 b=0		a=0 b=1.1	

3. In XY parallel view, parallel map 'mirror' onto the rectangle.
4. Rotate horizontally using the arrow keys so you are looking at the rectangle from behind.
5. Parallel map 'transp' onto the rectangle.
6. Select render settings LAMPLESS, SHADOWLESS or NORMAL
7. Set the background and environment colours to the same values.
8. Render

ANIMATION

Path animations

Path animations contain one major improvement: now the user can control over how the time is distributed along the path curve. This is done through the key editor window (Animate/Edit).

'V3samples\Path1.prj' example contains a simple 'move along a path' animation with a non-uniform knot-time distribution. To recreate a similar animation:

1. Create a sphere.
2. Select Animate/Edit to open the key editor.
3. Set the 'Method' selector in the key editor to 'Motion Path', make sure that the sphere is selected and click the 'Create' button. This opens the same window as the function Animate/Create/Path. Select a 'Polygonal' path and draw the curve on a view window.

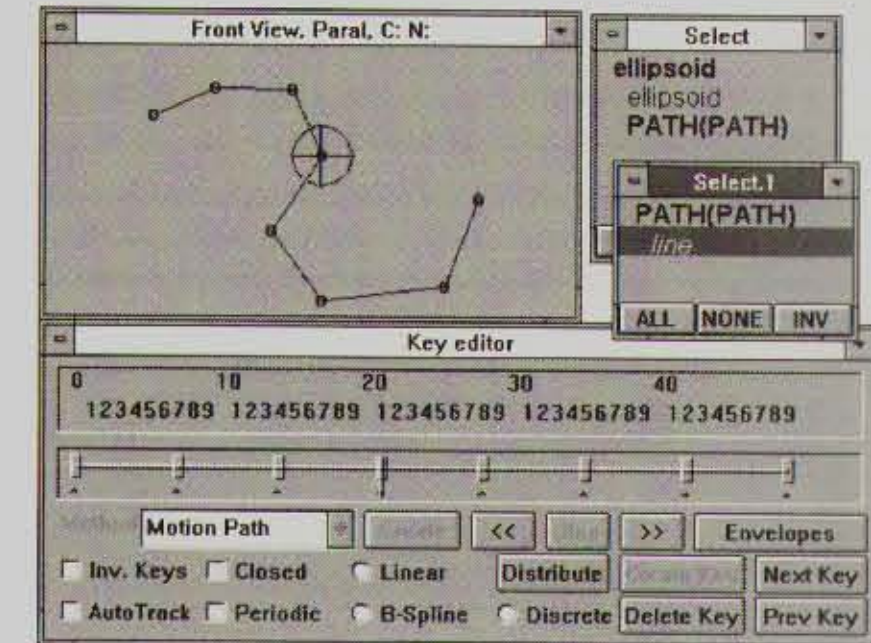


v3samples\
path1.prj



v3samples\
path1.prj

4. Use select window to make the created path method the current level. The time line of the path method displays as many knots as the defined path has points.
5. Play the animation to the end using the '>>' button of the key editor.
6. Pick some knots on the time line and move them to new frame numbers. For example, if you would like the object to reach the third point of the path at frame 20, just move the third knot to the frame 20. Then play the animation back to the beginning using the '<<<' button. Note how the sphere reaches each point of the line at given knot times.

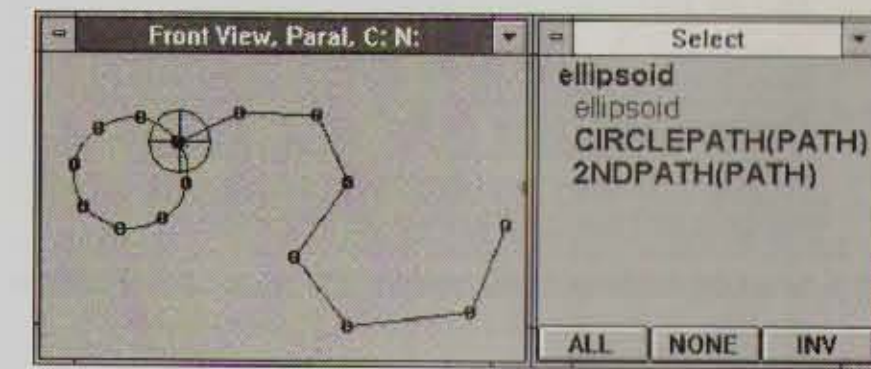


The animation methods recognize one new attribute 'Inherit'. This defines whether or not the time transformations applied by the method in question are inherited to successive methods.

The example 'V3samples\Path2.prj' demonstrates how two path methods can be used on the same hierarchy level. The 'Inherit' attribute of the first method is not set, so it is possible to modify the time line of the first method without affecting the behavior of the second one.



v3samples\
path2.prj





v3samples\
keyf1.prj

A simple keyframe animation

This example contains a simple keyframe animation. The animation consists of four keyframes. In the first frame the object is on the top left edge of the view window. In the second frame the object is in the middle of the window. In the third frame the object is rotated 90 degrees. In the last frame the object is moved to the bottom-right edge of the window and has doubled in size.

To create a key frame animation:

1. Select File\Project\New to clear all objects from the scene.
 2. Create a rectangle to the top left edge of a view window.
 3. Open the key editor window by selecting the menu Animate/Edit.
 4. Set the 'Method' selector to 'Keyframe'. Make sure that the rectangle is selected and click the 'Create' button. This initializes a keyframe animation.
 5. Press the 'Edit' button of the key editor. The select window will refresh and display the contents of the newly created keyframe method. You can see a copy of the rectangle and a coordsys in the select window. The copy of the rectangle stores the original state of the target, and the coordsys represents the first key frame.
 6. Click on the last frame number displayed on the key editor window. Time is changed to that frame.
 7. Click the 'Create Key' button. This creates a new key frame. Now select Modify/Linear/Move and move the rectangle to the right bottom corner of the view window. Then rotate the object and change its size.
- Note:** the correct items are automatically selected by the key editor, so don't select objects yourself while working with the key editor window unless you fully understand the way the keyframing system works.
8. Go to a new position in time by clicking the desired frame, click Create Key, modify and so on until all keyframes are defined.



v3samples\
keyf2.prj

Creating hierarchical animations using keyframing

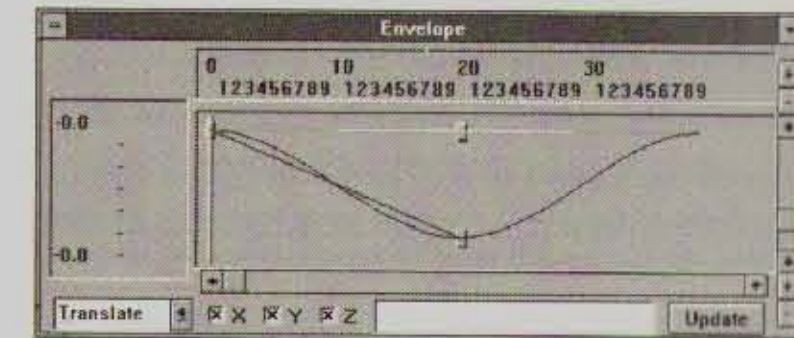
The basic idea of hierarchical animation is simple. For example, a bouncing ball can be constructed hierarchically so that the 'up-down' motion is defined first, and another motion moves the entire system forwards.

Also the keyframe animation method can be applied to all objects including those already animated by using any of the available animation techniques. A good example is an arm of a robot: the keyframe method can first be applied to the forearm. For example, the forearm can be rotated about the elbow. Then keyframing can be applied again to the complete arm in order to rotate it about the shoulder.

Let's create a simple bouncing ball animation first. In addition to hierarchical animation issue, this also demonstrates how to create closed loop motions.

1. Select the function Create/Visibles/Sphere and define a sphere near the top left edge of the view window.

2. Open the animation window and set 'Frames' to 40. Close the window.
3. Make sure that the sphere is selected (if not, select it using the select window), select 'Keyframe' from the method list of the key editor window and click the 'Create' button. This initializes a keyframe animation which consists of one keyframe.
4. Click the 'Edit' button. This prepares the key editor for editing the created keyframe method, and the select window is updated to show the hierarchy inside the keyframe method.
5. To keep this example short, let's create only two keyframes for the up-down motion. So, move to frame 20 by clicking on that number in the frame list at the top of the key editor, click on the 'Create Key' button and finally move sphere downwards so that it is near the bottom edge of the view window.
6. Set the 'Closed' checkbox of the key editor to get a closed motion, this means the sphere will return back to its initial position between frames 20 ... 39.
7. Use >> and << buttons to play the animation and to make sure it works as expected.
8. Let's get acquainted with the envelope window. Click the 'Envelope' button to open the envelope editor. Set Envelope to 'Translate' and check the X, Y and Z check boxes. You will see three curves which show you how the sphere moves in all three coordinate directions as a function of time. Note that these curves are measured in the object's own local coordinate space, not in the absolute space.



The envelope window shows how the object moves. If you are not satisfied with the motions, there are two ways of editing the animation: by modifying the key objects, or by modifying the envelope curves directly. Let's try both approaches. Leave the envelope window open so you can see how the envelope curves change when you move the key objects.

9. So, let's modify the second key frame object. Jump to the second key frame object by clicking the 'Next Key' or 'Prev Key' button (depending on what frame you are currently on). The keyframer automatically selects the correct objects; therefore you can apply modify functions without having to worry about object selection. So, select the function Modify/Linear/Move and move the sphere to a new location. The the curves in the envelope window will be moved accordingly.
10. Then modify any of the curves through the envelope window by clicking on one of the small rectangles denoting the knots of the curve and by moving it. If you want to see what impact this has to the sphere, click the 'Update' button. You can also multi-select several knots by dragging a box around the knots.

- 11. As a final step, we will apply keyframing again to the bouncing sphere object. Using the select window find the level object containing the original ellipsoid and the keyframe method. Select it and click the key editor's Create button to define a new keyframe method. Go to the last frame by clicking on it in the editor's frame scale. Click the Create Key button and move the bouncing sphere to the right edge of the view window. Play the animation and the bouncing sphere moves sideways while it bounces up and down.

Pivot points



v3samples\
keyf3.prj

As long as you only move objects in key frame animations, you don't have to worry about pivot points. As soon as rotation or scaling is applied, pivot points start to play an important role. The reason for this is that keyframer rotates and scales objects relative to the pivot point. For example, in a finger, pivot points should be located to the joints of the finger, because otherwise keyframe interpolation might pull joints apart from each other.

So, let's create a simple animated arm to demonstrate usage of pivot points. 1. First create a forearm. For the sake of simplicity, we'll use a rectangle to represent the forearm. Select Create/Visibles/Rectangle and define a suitably shaped rectangle.

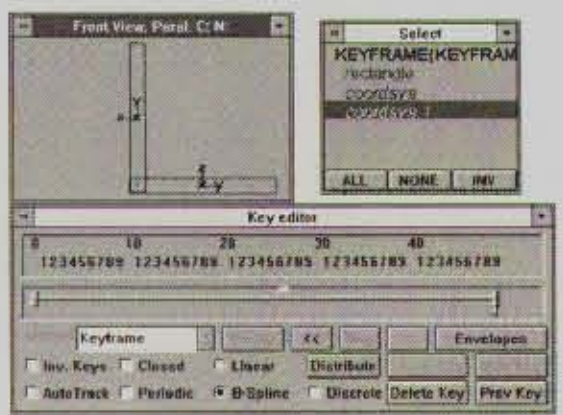
- 2. Open the key editor with Animate/Edit
- 3. Create a keyframe animation by setting the key editor's method selector to 'Keyframe' and clicking on the Create button. This initializes the animation with one keyframe.
- 4. Move to the last frame and click the Create Key button. This creates the second key frame.

Now its time to think of what kind of motion we would like the forearm to take. Let's rotate it around its elbow. At this point, pivots become important. Because we want the forearm to rotate around the elbow, the pivot point must be positioned at the elbow.

- 5. To redefine the position for the pivot point, just click the 'Pivot' button and set the new position by clicking in the View window where the pivot should be, in this case at the elbow.
- 6. Activate the Modify/Linear/Rotate function and rotate the forearm, say, 90 degrees about its elbow.

Note: the pivot point is actually the COG for the object so using Modify/About COGs/Rotate, Scale, etc would ensure any modifications are centered on the pivot point.

The rectangle is rotated 90 degrees about its elbow in the second key frame. The pivot point is marked with '+' sign.



- 7. Now play the animation back to the beginning (click the '<<<' button). The forearm rotates towards its initial position and the elbow is fixed to the pivot point.
- 8. Now the forearm is completely animated and it is time to create the upper arm. Make 'Root' object the current level and create another rectangle representing the upper arm. The hierarchical structure of the animation should look as follows:



- 9. Animating the upper arm doesn't differ from animating the forearm. This time we don't want to rotate the upperarm only, but the complete arm. So, select the 'Root' object and click key editor's 'Create' button. Then move to the last frame and click 'Create Key'.
- 10. Click the Pivot button and click on the shoulder, because we want the whole arm to be rotated at the shoulder.
- 11. Select Modify/Linear/Rotate and rotate about the shoulder.
- 12. Play the animation.



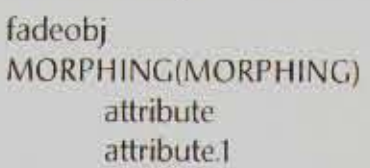
v3samples\
fade.prj

Animated fade-out

In this example we will see how to make an object fade during an animation.

- 1. Create an object. For example, a simple primitive such as a ellipsoid. For this example rename it as 'fadeobj'.
- 2. Use Modify/Properties/Fade function to define the initial fade level for the fadeobj. The fade value can be left at zero. It is necessary to open the window even though the default settings has been used as this gives the object an IFAD TAG. Unless this TAG is present the program won't be able to complete the animation correctly.
- 3. Select Create/Structure/Method. Select MORPHING from the method list and click OK.
- 4. Make the morphing method the current level.
- 5. Use Create/Controls/Attribute to create two attribute objects. These are simplest possible objects and will be used just to hold fade attribute values in a morph.

The hierarchy is now:



- 6. Select the first attribute, then the menu Modify/Properties/Fade and set fade=0. Click on OK.

7. Select the second attribute, select Modify/Properties/Fade and set fade=255 (maximum). Click on OK.
8. Select Animate/Edit. This opens the method editor. The editor displays the morphable object properties. This time we are going to morph only the IFAD tag which defines the fade attribute, so make sure Color, Materials, Geometry are not selected and that Tags button is.

Now the hierarchy is ready. If you render the animation, you will note that fadeobj fades out completely by the end of the animation. You may easily change the time interval during which the fade happens by setting the start and end time of the MORPHING method using Modify/Properties/Animation or by using the time line editor.

Skeletons



v3samples\
skelfide.prj

Real 3D V3 skeletons contain many new improvements which makes them an extremely powerful and easy to use tool for many different kinds of jobs. So, let's take a look at how the new skeleton system of Real 3D works.

Let's create a freeform leg and attach a skeleton to it. A good leg can be found in the objects directory as 'leg.obj', so load it in by using the function File/Objects/Insert.



A B-Spline leg.

The first step is to create a skeleton and attach it to the leg. This can be done by using the function Animate/Create/Skeleton. Select it from the menu.

Before you start drawing the skeleton, let's consider another new feature introduced by V3 called 'constraints'. Constraints consist of four angles associated with each joint of a skeleton. These angles define the minimum and maximum limits for the bones.

These constraints can be defined interactively while you are drawing the skeleton. So, let's try this now by creating a skeleton consisting of three bones: the first bone is short and represents the lower part of the spine. The two remaining bones represent the thigh and shin bones.

Make sure that the leg is selected, then click the first point near the sacrum and the second point over the hip bone. Now move the mouse towards the knee but don't fix the position for the knee yet. We will have to define proper constraints for the thigh bone first. So move the mouse 'forwards' so that it represents the knee in its uppermost position. Then move the mouse in the opposite direction so that it shows the position of the knee in its farthest back position. Then move the mouse back over the knee of the leg and fix the third point by clicking the mouse.



Define constraints by showing the frontmost and farthest back positions for the thigh bone.

Then move the mouse towards the ankle and repeat the steps above in order to define constraints for the shin bone as well. Then click the right mouse button, which terminates the skeleton creation.



A B-Spline leg and a skeleton.

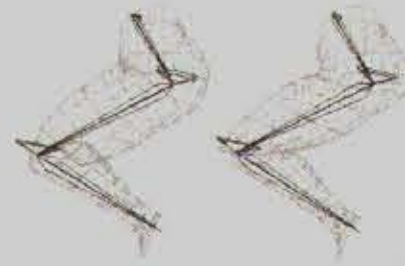
To make sure the skeleton is properly positioned inside the leg, take a look at it from different directions. If the skeleton doesn't match the leg properly, you can freely modify the skeleton at this point. Just find the skeleton primitive from the hierarchy and apply a suitable modification to it.

Now hit the 'u' key (or select the menu Animate/Control/Refresh). This attaches the skeleton to the leg so that when you later modify the skeleton, the leg will also be modified accordingly.

So, let's modify the skeleton primitive. When you created the skeleton by using the function Animate/Create/Skeleton, the proper object hierarchy was automatically set up for you. The skeleton primitive can be found under the 'SIMPLE SKELETON' level. So, select the 'skeleton' primitive, activate the function Modify/Special/Inverse Kinematic and move the ankle to a new position by first clicking near the ankle and then clicking at its new position. Hit the 'u' key and the leg is bent accordingly.

If you modify the skeleton to its limits, you can see that the leg may not look that good any longer because the skin near the joints doesn't behave as a real skin in the real world.

This problem can be solved by editing the so called 'fidelity' attribute of the bones. So, select the skeleton primitive and activate the function Modify/Properties/Skeleton Attrs. Go through all the joints and set fidelity to 40 %. Click Ok and refresh the animation system by hitting the 'u' key. The leg starts to look much better.

The effect of fidelity.

v3samples\
pointfid.prj

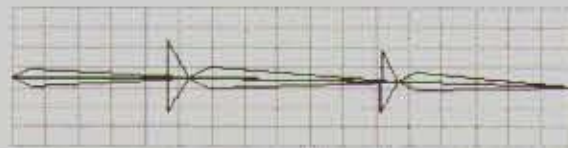
Fidelity

Because of the importance of the subject, let's go through another example demonstrating the use of the fidelity attribute. Get rid of the leg example by selecting the function File/Project/New.

In this example, we will create a rectangular polygonal mesh and attach a skeleton to it. When the skeleton is bent, it is very easy to observe what actually happens to the mesh and how the fidelity changes its behavior.

So, let's create a long and narrow horizontal mesh as shown in the figure below. Select the function Create/Build Freeform/Mesh and define the resolution as 18 by 8 points. To make the created mesh polygonal, select the function Modify/Freeform/Type and set the type to 'Polygon'.

Then select the function Animate/Create/Skeleton and draw a skeleton consisting of four points, as shown in the image below.

A polygonal mesh controlled by a skeleton.

Hit the 'u' key which causes the skeleton to be attached to the mesh.

Now find and select the skeleton primitive (can be found inside the 'SIMPLE SKELETON' level). Then bend it by applying Modify/Special/Inverse Kinematic to its last point. Hit 'u' to see how the mesh is modified.

Points near the joints cause problems.

Although we already know a way to solve problems related to points near the joints, let's do it once again so that we can analyze what actually happens to the mesh when the fidelity is changed. So, make sure the skeleton primitive is selected and activate the function Modify/Properties/Skeleton Attrs. Set the fidelity for all joints to 50 % and click Ok. Hit the 'u' key and you should see something like the following figure.



When fidelity is set to 0 %, position for each point is determined only by one bone of the skeleton i.e. all points attached to a certain bone are treated equally. The higher the fidelity the more the position of the point in question is also influenced by the next and the previous bone.

The 'fidelity' scroll bar in Modify/Properties/Skeleton Attrs. dialog allows you to define fidelity separately at each joint. But what if one needs more accurate control over fidelity? Say, one would like to define higher fidelity only for one point.

Real 3D also supports 'pointwise' fidelity i.e. the fidelity attribute associated with each bone can be overridden by a value associated with a target object. If the target object is a freeform mesh, a line or a triset, the fidelity attribute is actually associated with all points comprising the target object.

The check box 'Fidelity from targets' determines whether the fidelity is fetched from the target objects or from the skeleton.

Let's try this. Make sure the skeleton primitive is selected and activate the function Modify/Properties/Skeleton Attrs. Check the 'Fidelity from targets' check box and click Ok.

Refresh the animation system by hitting the 'u' key. This causes all the fidelity to be lost. The reason for this is that we switched the 'fidelity from target' option on and the default target fidelity is 0 %.

So, let's change the fidelity for all those points that are near the joints. Now, select the mesh and hold down the shift key while dragging points from it. When all the desired points have been selected, activate the function Modify/Freeform/Fidelity and enter the value 50 %.

Refresh the animation system by hitting the 'u' key and the fidelity effect is back.



v3samples\
skelbody.prj

Hierarchical skeletons

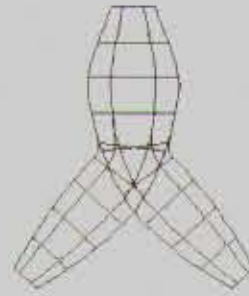
In this example, we will go through the idea behind hierarchical skeletons i.e. what they are and how they work in practise.

A hierarchical skeleton is a skeleton which contains one or more child skeletons. A good example of a hierarchical skeleton system is an arm and five sub skeletons, fingers.

Another example is a spine and its two sub skeletons, legs.

So, let's try this in practise by creating a simple body. We have created a suitable body just for this example. The body can be found in the objects directory as 'body.obj'. So, load it in by using the function File/Objects/Insert.

So, a 'body' object was loaded. Make the body the current working level by double clicking its name on a select window, and then multiselect both the meshes.



Then select the menu **Animate/Create/Skeleton** and draw a three point skeleton representing the spine as shown in the figure below.

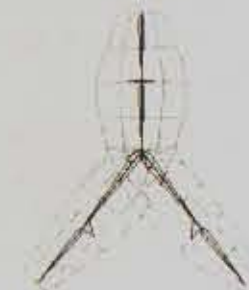
Note that we have to multiselect both the meshes. The reason for this is that then and only then the skeleton system treats the points of the freeform objects as individual objects. If you applied the 'Skeleton' function to the level containing the meshes, it would have been treated as a solid single object and the skeleton would have never been able to bend it.



Now it is time to create skeletons for the legs. So find the **SIMPLE SKELETON** method. It should contain one skeleton primitive i.e. the spine we just created. Select it and rename it as 'spine'.

Now, make the spine skeleton the current level by double clicking its name on a select window. In other words, sub skeletons are created inside the parent skeleton level just like meshes representing the body were created inside a level object named 'body'.

Then create two skeleton primitives by using the function **Create/Controls/Skeleton**. Both the skeletons should start from the second point of the spine and correspond the legs of the body. Use dragging to get an exact match and notice how constraints are defined relatively to the bones of the spine. Rename the skeletons as 'leftleg' and 'rightleg' to reflect their purpose.



A spine and two sub skeletons i.e. legs.

Now our hierarchical skeleton system is ready. Make sure that the skeleton matches properly the b-spline body by observing it from other directions. If so, refresh the animation system by hitting the 'u' key.

The skeleton is now attached to the body and we can try to modify the skeleton and see how the body behaves.

Note that in order to treat the hierarchical skeleton system as a whole, the parent skeleton must be selected. So, select the skeleton representing the spine and activate the function **Modify/Special/Inverse Kinematic**. Click near the ankle and move the mouse slowly down. When the leg gets its full stretch, it starts to pull down the spine and the other leg.

When done with inverse kinematics, hit the key 'u' and the body is modified accordingly.



v3samples/
skelmorp.prj

Morphing skeletons

Now that we have created a skeletonally controlled body, let's animate it by using morphing. If you have already deleted the body we created in the last example, it can be found as v3samples/skelbody.prj, so just load it in.

At this point, let's take a look at the hierarchical structure of the created object.

```
body
  level
    mesh1
    mesh2
    SIMPLE_SKELETON
      spine
        leftleg
        rightleg
```

Find the 'SIMPLE SKELETON' object and make it the current level.

Select the 'spine' skeleton and duplicate it by using the function **Modify/Structure/Duplicate**.

Now modify the duplicated skeleton by using the function **Modify/Special/Inverse Kinematic**.

Create a third key object by duplicating the modified skeleton and again use inverse kinematics to modify the last duplicated skeleton.



Three key skeletons

Select all the skeletons inside **SIMPLE SKELETON** method and activate the function **Animate/Create/Morphing**.

The object hierarchy should now look as follows:

```
body
  level
    mesh1
    mesh2
    SIMPLE_SKELETON
      spine
        leftleg
        rightleg
      MORPHING
        spine1
          leftleg
          rightleg
        spine2
          leftleg
          rightleg
        spine3
          leftleg
          rightleg
```

Play the animation and, as you can see, the skeleton is bending through the key skeletons you created, and the actual body is bending with it.

Surface method



v3samples/
rollcube.prj

The following examples demonstrate the new powerful 'surface' method. This method makes rotating walking etc. objects automatically travel along surfaces.

The hierarchy syntax of a surface method looks like the following:

```
Root
  target
    SURFACE(SURFACE)
      gravity
      surface
      links
        link1
        link2
        ...
```

This perhaps looks complicated, but is not that difficult to use. Let's make a cube roll along a table.

1. Start a new project and create a new level called 'rotator'. Make it the current level.
2. Take a front view (View/Viewcam/Set XY). Create a cube.

3. Select Create/Controls/Offset. Drag a box around the top left corner to create an offset exactly at that corner. Duplicate the created offset and move it to the top right corner. Create two more copies and place them to the remaining two corners of the cube.
4. Multi-select the four offsets and select Create/Structure/Link; four links are created. We will need them later in another part of the hierarchy, so select Modify/Structure/Cut to cut them to the clip buffer.
5. Make root the current level. Select Create/Structure/Method and pick 'SURFACE' from the method list. Then click OK.
6. Make the created surface method the current level. As we see in the hierarchy diagram above, the first parameter defines the direction of the gravity. Select the menu Create/Controls/Axis and draw a vertical line pointing downwards. The length and position of it does not matter, only the direction.
7. Next we create the surface on which the cube travels. Take a top view, select Create/Visibles/Rectangle and draw a rectangle. Select Modify/ Properties/Attributes, check Infinite and click OK. Now the cube can safely continue rotating forever.
8. Take a front view and move the rectangle below the cube if it is not there yet.
9. Create a new level using Create/Structure/Level. Make it the current level.
10. Select Modify/Structure/Paste. The links which you cut in the step 4 above are pasted back to the hierarchy.
11. Now the surface method is finished. There is one more thing to do: the cube itself is still static. Go to the root level, select Animate/Create/ Rotation, select the 'Normal' and 'Periodic' options and click OK. Draw the rotation coordsys in the middle of the cube.
12. Find the rotation method from the hierarchy (it is inside the level 'rotator') and select it. Pick the menu Modify/Properties/Animation, check Frequency and enter a value of 5. Click OK.
13. Play the animation. The cube makes 5 revolutions and travels quite far away.

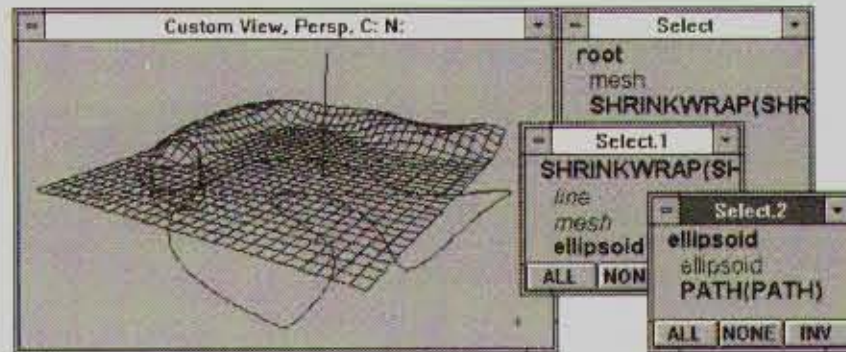
Other examples of this subject are:

- v3samples\surface1.prj A walking robot was created by using the sweep method. The surface method causes the walking robot actually move because of the friction between the floor and the feet of the robot. The friction points are placed into the feet of the robot.
- v3samples\surface2.prj Another surface method example. A walking robot goes upstairs. NOTE: Before playing this animation, set File/Windows/Animation/Frames to at least 120.
- v3samples\surface3.prj This is another 'place walking robot on the floor' example. However, one extra sweep method rotates the robot causing the robot to wander to different directions as if he couldn't make up his mind about the direction.
- v3samples\surface4.prj This one is quite crazy; a rotating 'thing' on a floor.
- v3samples\surface5.prj In this example, a walking robot was placed on a non-planar B-Spline surface.

Shrink wrapping

This example demonstrates the 'shrink wrapping' animation method. A sphere moves under a mesh deforming it.

1. We start the example by creating a sphere (Create/Visibles/Sphere).
2. Take a top view. Select the sphere and the menu **Animate/Create/Path**. Select for example the B-Spline, Closed, Periodic and Auto phase options. Click OK, and draw a path for the sphere on a view window.
3. Select **Create/Build Freeform/Mesh**. When the program asks the point count of the mesh, enter 20 * 20 points and click OK. Then shape a rectangular mesh covering most of the sphere and its path.
4. Take a front view and lift (Modify/Linear/Move) the mesh above the sphere.
5. Multi-select the animated ellipsoid and the mesh, and then pick **Animate/ Create/Shrink wrapping** from the menu. A dialog is opened; select 'Parallel' and 'Cumulative' options. Click OK. Now the program waits you to draw a line on a view window. The line will define the amount and direction of shrinking. Click in the middle of the mesh and draw a line downwards to the level of the middle point of the sphere.
6. Make sure that either undo is on or save the project at this point. Then play the animation. Animated, cumulative shrink wrapping cannot be played backwards to the original state, the shape changes are permanent.



Animated shrink wrapping



v3samples\
shrink1.prj



v3samples\
shrink2.prj

7. Use undo to restore the animation after the playback or reload it from disk. Let's experiment with non-cumulative shrink wrapping, too. Select the SHRINK WRAPPING method object from the hierarchy. Then select **Animate/Edit** and switch the cumulative option off. Replay the animation.

When using shrink wrapping, it is important to ensure that the density of the mesh to be shrunk corresponds the level of detail of the target object. Otherwise small objects may get through the mesh. A good trick is to surround the target object (a sphere in the example above) with another, bigger one, and make it RT-invisible so that it does not affect rendered images.

2. MENU FUNCTIONS

File/

Objects/Load

The function **File/Objects/Load** can also load 3D Studio files. The imported data is represented as triset primitives. Triset shading can be controlled with the **Modify/Freeform/Type** function.

Project/Insert Sections, Save Sections, Replace Sections

The sections dialog contains three new checkboxes 'Landscapes', 'Trees' and 'Rendering settings'. These controls can be used for saving and loading fractal and render settings libraries.

Materials/Window

Edge X/Y

Edge X and Y buttons give additional control of the texture gradient feature. Normally the **Grade X/Y** function interpolates the texture map color to the object color at the edges of the texture map. If **Edge X** is set, the gradient is computed inside the texture map, but the vertical edge appears sharp. **Edge Y** does the same to the top and bottom edges of the texture.

Freq X and Y

In previous versions of Real 3D, setting a non-zero Frequency value for a texture map made the texture appear the given amount of times inside the mapping, i.e. it made the texture map denser. This denser tiling was always finite.

Now you can enter a negative frequency value, which works in a similar way, to define the amount of repetitions inside the mapping geometry by taking the absolute value of the fields, but infinite tiling is used in this case. This has one immediate advantage: if, for example, a spherical mapping is used with a negative, non-default frequency, and either bump mapping or texture gradient is also used, then the 'seam' where the mapping reaches the full 360 degrees disappears, because the tiled texture sequence is repeated again over the seam.

Texture antialiasing

The texture antialiasing feature improves rendering quality, for example, when rendering infinite, chequered planes so the texture gets dense at the horizon where bad aliasing problems could arise.

Glow

The material editor now contains two new material properties: glow size and glow brightness. As the name indicates, these properties may be used to simulate glowing materials like fire, laser beams and coronas.

Glow size controls the width of the glowing area around the edge of the material. Glow brightness defines the intensity of the glow.

Glow is computed as a post processing effect. Therefore, in order to activate glow processing, you must use the Post effect editor of the Render Settings dialog to insert a 'Glow' effect to the post effect list. For further details, see the chapter 'Post effects'.

The RPL variables to control glow are 'gs' for glow size and 'gb' for glow brightness. Their values can be between 0 and 100. The variables can be set in Scope handler formulas and RPL procedures.

Depth scope handler

The 'Depth' scope handler limits the material scope in the 'z' (depth) direction to the interval specified by the 'a' and 'b' scope handler parameters. For example, if $a=0$ and $b=0.1$, a parallel mapping maps the material 0.1 abs. space meters deep into the target objects from the mapping rectangle level in the direction specified by the small peak. This handler may be used for example to map a texture onto one side of a cube only.

Angle scope handler

Another new scope handler is 'Angle'. It defines the scope of the material as a function of the viewing angle to the surface. The angle is described with a value between -1 and 1. The values -1 and 1 correspond to directions perpendicular to the viewed surface, and 0 corresponds to viewing along the surface.

The parameters 'a' and 'b' specify the range, inside which the scope is unchanged. Outside the interval, scope is set to zero. Typically, this handler can be used to create 'one-sided' textures. For example, a texture map on a rectangle can be set to paint only one side of the rectangle by selecting the Angle handler with $a=0$, $b=1.1$ (the extra 0.1 is to eliminate accuracy problems). If a is set to -1.1, and b is set to 0, then the opposite side will become textured.

Roughness bump handler

The roughness material property has also been removed. Instead, there is now a more flexible bump handler called 'Roughness' available.

The parameter 'a' defines the density of bumps on the surface. The smaller the value, the less bumps you get, and small values like 0.1 produce small amount of isolated bumps. If 'a' is left at zero, Real 3D uses a default value 1.0 automatically. This value creates a random bump on every position on the surface.

The parameter 'b' defines the height of the bumps. The default value, used when 'b' is left at zero, is 1.0.

Note: because of the removal of the original roughness parameter, the RPL variable 'ro' is no longer available.

Dither color handler

The Dither material property slider has been removed. It has been replaced by a more versatile color handler called 'Dither'.

The 'a' parameter defines the density of random color spots on the surface. The smaller the value, the less spots you get, and small values like 0.1 produce quite isolated spots. If 'a' is left at zero, Real 3D uses a default value of 1.0 automatically. This value creates a random spot on every position on the surface.

The 'b' parameter defines the intensity of the spots. The color of a spot is obtained by modifying the brightness of the original color randomly. The default value, which is used when 'b' is left at zero, is 1.0.

Note: because of the removal of the original Dither property, the RPL variable 'di' is no longer available.

Material color

A new color handler called 'Col.repl' is included. This handler uses values entered into the materials window transparent RGB fields to specify the color of a material. In other words, the color replaces any other color definition like object's color or color from a texture map.

For example, you may create a specular, reflective material called 'gold'. If you apply it onto a red object, the result is something shiny but red. If you use FromTran handler and define Transparent RGB to be yellow, the material makes the target objects yellow, which looks more like gold.

Relative texture coordinates

Relative bitmap-size independent texture coordinates have been implemented. They can be accessed from RPL and formula handlers with variables 'relx', 'rely' and 'relz'.

Relative coordinates have the following ranges:

Parallel mapping:

The Horizontal mapping edge corresponds to an interval of [0,1] in relx. The Vertical mapping edge corresponds to an interval of [0,1] in rely. Mapping plane normal whose length is the average of mapping edges corresponds to an interval of [0,1] in relz.

Cylinder mapping:

Circumference of the mapping cylinder corresponds to an interval of $[0, 2\pi]$ in relx. Axis corresponds to an interval of [0,1] in rely. Radial distance from axis to circumference corresponds to an interval of [0,1] in relz.

Sphere mapping:

Circumference of the mapping sphere corresponds to an interval of $[0, 2\pi]$ in relx. The arc between the poles corresponds to an interval of $[0, \pi]$ in rely. Radial distance from center to circumference corresponds to an interval of [0,1] in relz.

Disk mapping:

Circumference of the mapping disk corresponds to an interval of $[0, 2\pi]$ in relx. Radial distance from the center to the circumference corresponds to an interval of [0,1] in rely. The mapping plane normal whose length is the radius of the disk corresponds to an interval of [0,1] in relz.

Default mapping:

relx, rely, and relz are the absolute space coordinates.

Spline mapping:

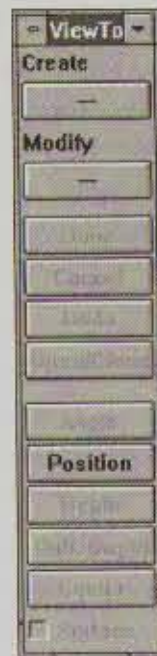
relx and rely are the 'natural' surface parameters: the edges of the mesh correspond to an interval of [0,1] in relx and rely. relx is always zero.

New image formats

It is now possible to use PPM and JPEG images as texture maps.

Materials/Purge

This function scans the object hierarchy for SMAT tags (material references) and any materials not appearing after the scan are deleted. This means that any defined materials which are not currently in use by the project are removed. This is a convenient way to clean up a large project which has too many accumulated, and possibly duplicate extraneous materials.

Windows/ViewTool

This window can be opened by selecting File/Windows/ViewTool from the menu.

The window consists of a number of buttons which can be used to control creation and modification tools, such as 'Create/Visibles/Cube'.

The main purpose of this window is to allow faster access to the functions and provide greater functionality in the editing environment.

Buttons:

Create

Displays the next buffered action if it is the creation of a primitive found in the Create/Visibles menu. Clicking on this button opens a floating menu which duplicates the Create/Visibles menu.

Note: Implemented in Windows version only.

Modify

Displays the next buffered action if it is a modification as found in the Modify/Linear menu. Clicking on this button opens a floating menu which duplicates the Modify/Linear menu.

Note: Implemented in Windows version only.

Done

Completes the current phase of the creation process. For example, when creating a polymid, press this button, click on this button when all the points of the base polygon have been defined.

Cancel

Cancels the current creation/modification operation.

Undo

Removes the last point.

Open/Close

When creating a line or b-spline curve, this button toggles the curve between open and closed.

Angle

During the execution of Modify/Linear/Rotate, this button can be used to enter the angle (in degrees) of rotation. The button only becomes active after the rotational axis has been defined. When creating sector primitives, the open angle (in degrees) of the sector can be entered numerically. The button only becomes active when the first two points of the creation have been defined.

Position

Can be used to enter the coordinates of a point numerically.

Depth

When creating a primitive with depth (e.g. cube), this button opens a dialog where the depth can be entered in metres. This button is only active during the creation of a suitable primitive.

Outl. Depth

When creating a primitive which has a depth, this button allows the depth to be defined visually by drawing a line in a view window. This button is only active during the creation of a suitable primitive.

Center

This button places the next point in at the center of the points defined already in a creation process. For example, the second end of a culcone can be defined to be centered over the first.

Example 1: Create a cube with a depth of 0.1:

- Select the function Create/Visibles/Cube
- Start cube definition by clicking once in a view window
- Click the 'Depth' button of the ViewTool window and enter the desired depth of the cube
- Complete the cube definition by entering the second point on a view window.

Example 2: Create a cone with a depth equal to the width of the previously created cube.

- Select Create/Visibles/Cone
- Define the center and radius in a view window.
- Click the 'Outl.Depth' button of the ViewTool window. This allows you to define the depth of the cone by entering two points in a view window. The distance between defined points will be used as a depth for the primitive to be created. So, drag two points from the previously created cube.
- Define the apex for the cone with a left mouse button click.

Example 3: Create a cylinder sector of 45 degrees:

- Select Create/Sectors/Cylinder
- Define center and first angle on a view window.
- Click the 'Angle' button of the ViewTool Window and enter the desired angle (45) degrees.

Example 4: Rotate objects 45 degrees:

- Select objects to be rotated
- Select the function Modify/Linear/Rotate
- Start rotating objects as usual by defining two points by clicking in any view window.
- Click the 'Angle' button of the ViewTool window and enter the desired rotation angle in degrees.

Example 5: Create a cone whose apex is positioned exactly at the center of the base circle.

- Select Create/Visibles/Cone
- Define the center and radius by clicking two points in a view window.
- Click the 'Center' button of the ViewTool window.

Example 6: Move objects exactly -0.5 space units along the 'x' axis.

- Select objects to be moved
- Select the function Modify/Linear/Move
- Start moving objects by clicking a point on a view window
- Click the 'Position' button of the ViewTool window. This opens a dialog which shows you the coordinates of the last clicked point. for example:

X: 0.43 Y: -0.03233 Z: 0

Add '-0.5' to the X: field

X: 0.43-0.5

and click OK.

Example 7: Create a cone whose apex is located precisely at the point (1,1,1):

- Select Create/Visibles/Cone
- Define the base circle as usual
- Click the 'Position' button and enter coordinates x=1, y=1, z=1.

Create/

Structure/Link

See Structure/Group below.

Structure/Group

The system of how groups and links find their targets is based on the tag 'SIDE'. The target of a link reference is determined by finding the first object with the same SIDE tag identifier, then searching from a minimal sub hierarchy containing both the link and the target. In other words, the target is first searched for in the same hierarchy level, then in its parent level and so on.

Duplicate identifiers usually don't cause problems because the object hierarchy determines how the references are resolved. For example, the RightLeg could be created by duplicating the LeftLeg:

```
Legs
  LeftLeg
    mesh SIDE=leg
    group SIDE=leg
  RightLeg
    mesh SIDE=leg
    group SIDE=leg
```

Problems may arise if a link and its target are duplicated into the same level as the originals:

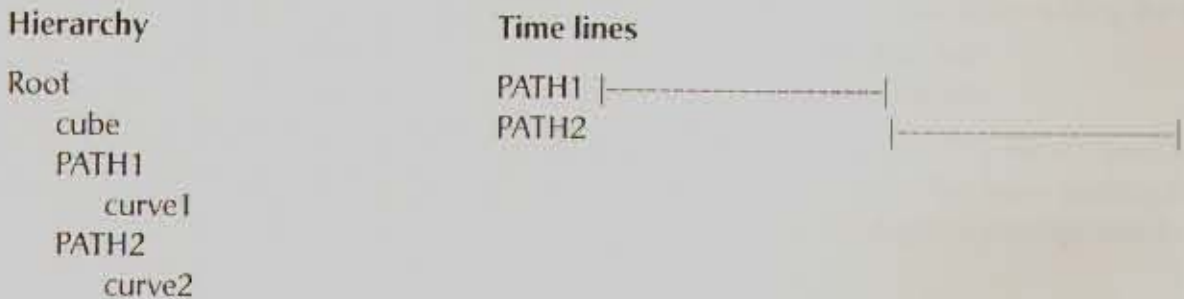
```
Legs
  mesh1 SIDE=leg
  group1 SIDE=leg
  mesh2 SIDE=leg
  group2 SIDE=leg
```

In this case, both group1 and group2 refer to mesh1, although the intention has clearly been that group2 refers to mesh2.

If a situation arises where a link/group points to another target as intended, the problem can be solved by defining a new, unique SIDE tag for both the target and the link: select the link, then menu Modify/Properties/Tags, select the SIDE tag and change it, for example, to 'SIDE=noproblem', click OK, and then do the same for the target.

Structure/Method

The Create/Structure/Method dialog contains a new option 'Inherit', which is set by default. If the option is unselected, time modifications defined by the method are not inherited to successor methods, but are visible only inside the method. For example:



If the Inherit option is set and you modify the time line of the PATH1, PATH2 will be affected as well. If you don't want this effect, just reset the Inherit attribute for PATH1. Then you may modify the time line, phase, frequency etc. attributes of PATH1 freely without changing the behavior of PATH2 in any way.

The time inheritance feature is controlled by the tag IINH, which can be associated with all animation methods. If the value of this tag is 0, time isn't inherited. If the tag is not zero or missing, time is inherited.

For a list of new and changed methods, see the chapter 3 about the Animation system.

Controls/Helix

Angles are now used for all 'high level' functions, such as Create/Controls/Helix. However, radians are still used internally, and all functions which directly reflect internal data structures (such as Modify/Properties/Tags) deal with radians.

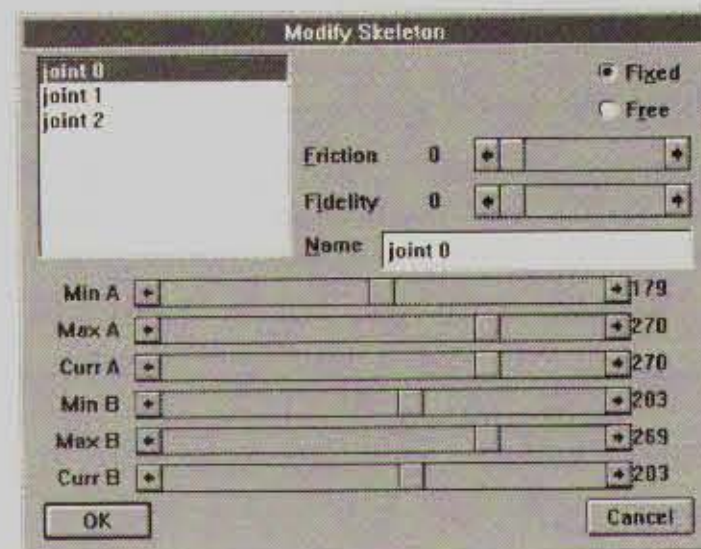
Controls/Skeleton

A skeleton is a special purpose primitive, which can be used to control other objects in combination with the skeleton method.

A skeleton creation is started by selecting the Controls/Skeleton menu. It is defined by clicking the positions of desired amount of joints in a view window; a right mouse button click finishes the creation. During skeleton creation, initial angle constraints of joints are also defined. An angle constraint appears as a triangle at the beginning of the bone; the bone can be bent only so far that the previous bone stays inside the sector defined by the triangle. The angle constraints of the first bone can't relate to a previous bone, so they are defined in absolute space, unless you are creating a sub skeleton. In this case, the constraints are defined in relation to the end of the nearest bone end of the parent skeleton.

When the skeleton has been defined, the skeleton attributes dialog is opened.

The skeleton attributes dialog



The dialog includes controls for joint names, frictions, fidelity, angle constraints and bone orientation. The controls are:

Skeleton type

If the type of a skeleton is 'Free', the whole skeleton will be moved if inverse kinematic requires the skeleton to reach further than its length allows.

Name

The dialog displays a list of joints. When you select a joint from this list, its name appears in the name edit box. You can rename a joint, for example, to call a middle joint of a 3 point skeleton 'elbow'. This helps to identify it quickly later. By default, the first joint (the point you clicked first when drawing the skeleton) is called joint 0. The end point of the skeleton has the highest ordinal number.

Friction

You can define a friction value for each joint of a skeleton primitive. If the value is zero in all joints and you pull the skeleton from the end point, the last segments of the skeleton are affected first, and inverse kinematics modification uses a minimal amount of segments to relocate the end point of the skeleton. If the value is greater than zero in all joints, all the skeleton parts become affected immediately. The maximal friction value 100 makes a joint rigid: the joint bends only if bending in other joints isn't enough for relocating the point modified by inverse kinematics. The friction is meaningless in three point skeletons, because there is only one point where the skeleton bends.

Fidelity

Fidelity can be used to control the orientation of skeleton targets with respect to the direction of the skeleton. The value of 0 causes the orientation of the target object to exactly match the orientation of the corresponding bone of the skeleton. The higher the value, the more the orientation is determined also by the next and previous bones of the skeleton. This can be used to simulate skin behaviour, for example, when using skeletons for character animations.

Note: the fidelity value of the last joint doesn't have any effect, because there is no bone starting from that joint.

Another way to control fidelity is to use the FSKF tag. FSKF defines the same 'fidelity' property as the value in the skeleton attributes dialog, but the tag defines fidelity for each target separately, thus allowing more accurate control. The value range of the tag is from 0 to 1, 0 corresponds to a situation where the orientation of the target follows the bone totally.

For example, a spline-modelled arm, subdivided into cross-sections (point groups) which are controlled by a SKELETON method can be made to bend smoothly at the elbow by adding the tag 'FSKF=0.5' to the cross-sections at both sides of the elbow joint.

Angle Constraints

Four angles are used to define constraints for each joint. **MinA** and **MaxA** define the minimum and maximum angles of a sector in the plane in which the skeleton was created; the bone starting from the selected joint can bend only inside the defined sector. The angle of 180 degrees corresponds to the direction of the bone which ends at the selected joint.

MinB and **MaxB** define another sector which is perpendicular to the MinA-MaxA sector. Together they define a rectangular space cone inside which the bone can move.

The constraint angles are defined interactively when the skeleton is created, and they may be adjusted later with the sliders in the skeleton attributes dialog.

Note: MinB-MaxB sector can't be defined during the interactive creation process, and therefore MinB and MaxB are both set to 180 degrees by default. This allows the skeleton to bend only in the plane in which it was created.

Sub skeletons

Skeletons can contain sub skeletons. When a sub skeleton is modified by inverse kinematics, the parent skeleton is also affected.

Hierarchical skeletons can be created exactly the same way as non-hierarchical skeletons: just set up the hierarchy, and when the animation is played for the first time, the target objects are automatically fixed to the skeleton.

Sub skeletons can be attached to any joint of the parent skeleton and the number of sub skeletons isn't restricted in any way. A hierarchical skeleton system is 'glued' together automatically when the animation is refreshed/played for the first time. The sub skeleton is attached to the nearest joint of the parent skeleton.

Note: the sub skeleton isn't pulled to the joint of the parent skeleton but remembers its original displacement. If an exact match is desired, use dragging to position the first point of the sub skeleton on the desired joint of the parent skeleton.

If the type of the sub skeleton is 'Fixed', the skeleton will always remain attached to the parent skeleton. If the sub skeleton is 'Free', it will be pulled away from the parent skeleton if the parent is fixed and the inverse kinematic requires the sub skeleton to reach further than its length allows. e.g. an arm can be made to leave the body in order to reach an object.

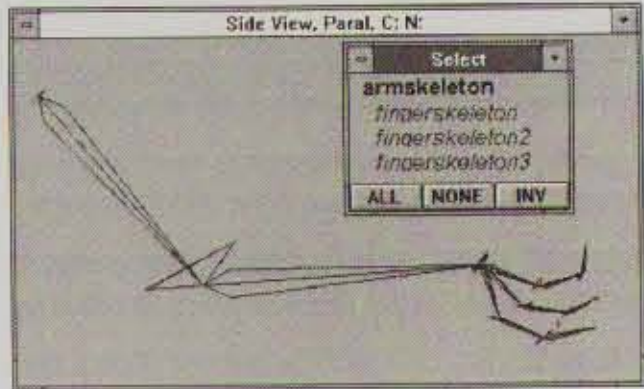
To modify hierarchical skeleton systems, the uppermost skeleton (root skeleton) must be selected.

When creating sub-skeletons, the constraints are automatically defined relative to the parent skeleton.

Example: Create a hand

- Create a skeleton to represent a shoulder, elbow and wrist.
- Make the created skeleton the current level (double click the name of the skeleton in a select window).
- Create five sub skeletons to represent the fingers. Note how constraints for the first joint are defined in relation to the parent skeleton.
- Select the 'shoulder-elbow-wrist' skeleton (the parent skeleton of fingers) and apply Modify/Special/Inverse Kinematics to the wrist.

A hierarchical skeleton.



Fractals/Landscape

Settings defined in the Landscape dialog can now be saved and loaded. The system is similar to material or grid handling. In other words, You can create a list or a 'library' of fractal settings. The File/Project/Save Sections dialog contains a checkbox 'Landscape', which can be used to save landscape libraries separately to disk. File/Projects/Insert, Replace and Save automatically load and save the contents of landscape libraries.

To create a collection of fractal settings:

Use the Fractals/Landscape function to experiment with parameters. When you get a suitable result, enter a name into the name edit box of the landscape dialog and press the Save button. This adds a new entry to the memory resident landscape library. Then continue experimenting, and whenever you get a good result, give it a new name and hit Create to store the parameters.

You can retrieve settings of a previously created landscape by pressing the Load button and by selecting the desired entry from the list. The Delete button allows you to remove entries from the landscape library.

When an appropriate landscape collection has been defined this way, select the menu File/Project/Save sections, clear all sections except 'Landscapes' from the section selector, click OK, then define a suitable file name to store' the collection with the file selector.

You can reload a landscape collection with, for example, the File/Project/Insert Sections function. Remember to check 'Landscapes' when using this function.

Fractals/Tree

The procedures to create, save and load fractal tree collections is similar to fractal landscape handling as described in the Fractals/Landscape section, but use the Fractals/Trees menu for tree creation, and when loading and saving tree libraries, use the Fractal Trees box in the project load and save sections selectors.

Modify/

Properties/Attributes

The Modify/Attributes dialog has a new checkbox 'Bounding Box'. Checking it replaces the wire frame representation of selected objects with a bounding box, which draws much faster. If the selected object is a level, then a single bounding box which surrounds the entire level is created.

Properties/Animation

See Create/Structure/Method for information about new attributes.

Properties/Skeleton Attr.

See Create/Controls/Skeleton menu in this appendix for a detailed description of skeleton attributes.

Properties/Geometry

This function opens a dialog which can be used to view and edit the geometric properties of primitives. Absolute space units (meters) and degrees are used.

The geometry dialog



To use this function, select primitives and select this menu. This opens a dialog displaying a list of the geometric attributes of the first selected primitive. You can select the attribute you want to modify from the list and change its value using the XYZ fields. Clicking OK confirms the modification and the function proceeds to the next selected primitive. Pressing 'SKIP' ignores modifications of the currently displayed primitive and the function proceeds to the next primitive. CANCEL stops the function.

When a primitive type which may include a variable amount of point data is modified, the dialog displays either one 'U' slider or two 'U' and 'V' sliders. With these sliders, you can select the point to be modified.

Example: to move a sphere to a new position, say $x=0.5$, $y=0$, $z=0$:

- Select the sphere
- Select the function Modify/Properties/Geometry
- Select the attribute 'Center' from the attribute list and enter the desired coordinates for the center point.

The primitive type specific attributes which can be modified with this function, are:

Rectangle:

- depth - 'dvect' vector (direction and size)
- $p0, p1, p2$ - three points defining a rectangle

Cube:

- depth - extrusion vector
- $p0, p1, p2$ - three points defining the base

Pyramid:

- apex - position for apex
- $p0, p1, p2$ - three points defining the base of the pyramid

Ellipse:

- depth - 'dvect' vector
- center - center
- a, b - axes

Cone:

- apex - the top of the cone
- a, b, c - axes
- p, m, n - point p and two vectors m, n define a plane which cuts the infinite cone to a finite solid (m cross product n is the normal vector of this plane).

Ellipsoid:

- center - center point
- a, b, c - axes. If these are perpendicular to each other and have equal lengths, the shape is a sphere.

Hyperboloid, Cutcone, Cylinder, Ellipse:

- center - center point
- a, b, c - axes defining a quadric
- $p1, m1, n1, p2, m2, n2$ - two planes defining the end faces.

Offset, Aimpoint:

- position - clear

Viewpoint:

- position - position of the camera
- horiz - horizontal 'tilt' direction vector
- direction - camera direction.

Coord.sys:

- origin - position
- a, b, c - axes

Line:

- $p0, p1, \dots, pn$ - the points of the line

Spline:

- $p[0,0], p[0,1], \dots, p[n,n]$ - control polygon points

Polygon:

- stick - 'dvect' boolean volume direction
- $p[0], p[1], \dots, p[n]$ - the shape

Polyhedron:

- depth - Extrusion vector
- $p[0], p[1], \dots, p[n]$ - the base shape

Polymid:

apex – the top
p[0], p[1], ... p[n] – the base shape

Cutpolymid:

base[0], base[1], ... base[n] – base points
top[0], top[1], ... top[n] – cover points

Triset:

p0, p1, ... pn – the points of the mesh

Properties/Fade

Real 3D's Rendering engine now recognizes a new object attribute 'Fade'. This attribute can be defined with the Modify/Properties/Fade menu or with the Tag editor. The attribute is stored as an integer tag '1FAD nnn' where nnn is in the range between 0 and 255.

The Fade attribute makes an object translucent. The minimum value 0 doesn't affect the object at all, whereas the maximum value 255 makes the object completely disappear from the rendered image, as if it did not exist at all. Also, the object's shadows and reflections are faded.

The Fade attribute may be used for example to easily fade objects in and out in animations, by morphing the tag value. Using the fade attribute is different from using transparent materials:

- Translucency created by fade renders much faster than real 'optical' transparency.
- Fade has no refractions or reflections.
- 100 % full transparency can also be achieved with colored objects.

Special/Reflect

This function moves selected objects so their reflections will be seen on a surface at a position on the object surface pointed to with the mouse.

For example, when you want to move a light source to where it's specular highlight will appear in a certain position, select the light source, then select this function and click in the position where you want to have the center of the highlight. If the click doesn't hit any object, the light source will not move.

Freeform/Type

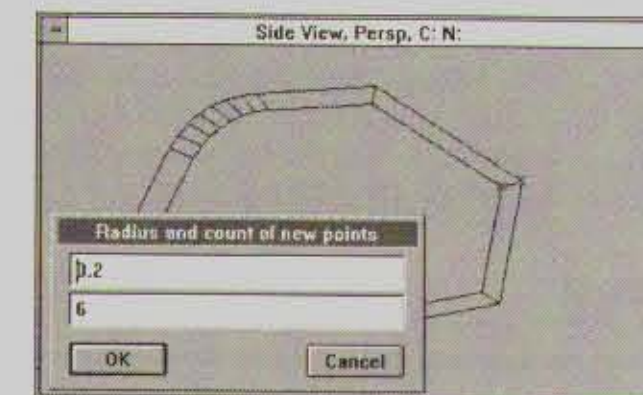
This is a useful feature when handling converted (imported) objects. The Modify/Freeform/Type dialog contains a 'Clockwise normals' option, which may sometimes produce better shading than the default automatic orientation method, which tries to set the face normal directions by checking the adjacent face normals. Which method is better depends on the modeller which created the object and the conversion software used.

Freeform/Round

The function Modify/Freeform/Round can be used to round polygonal lines, b-spline curves, polygons, polyhedrons, polymids and cutpolymids.

To use this function, select the desired points by placing them on the vector stack (drag around them while holding down the shift key). Then select the function Modify/Freeform/Round and define the rounding radius and the number of points to be used for rounding.

Rounding a polyhedron



Freeform/Convert to Triset

The Convert to Triset function converts the selected objects to trisets, which can then be exported to 3DS for example. Unlike the old 'Triset from mesh' function, this one is based on the real shape (B-Spline, quadric etc.) and can handle all kinds of objects.

Note: Boolean Operated objects are not yet supported. They are exported as if they were not operated at all.

Freeform/Surface to Groups

This function can be used to subdivide a mesh or a line to suitable sub groups. To use the function, select freeforms and then this function from the menu. Enter the size of groups to the U and V fields, and click OK.

When you modify lines with this function, the 'V' size is not used.

For example, you can subdivide a 8*8 pixel mesh to sixteen sub groups of 2*2 points each.

Freeform/Fidelity

Skin behaviour of freeforms controlled by skeletons can be defined using so called fidelity attribute, which can be attached both to skeleton joints and to skeleton targets. See the description of skeletons for details.

Freeform/Fidelity function defines the fidelity attribute for the selected points of the selected freeforms. To use the function, select some freeforms, then Shift-Drag some points (you can also use Lasso selector) and select this function from the menu. Define the fidelity value and click OK.

View/

Type/Surface

When this mode is set, 3D coordinates are automatically defined by mouse input. The mouse position is actually moving along object surfaces instead of along the input plane. This function is very powerful. For example, the Type/Surface function makes it easy to position objects accurately on freeform surfaces or blend B-Spline surfaces.

Camera/Camera View

Camera view is now updated interactively when you move camera objects provided the View/Camera/Camera View option is set and the window with camera view isn't the active one.

Previously, camera view was updated only during animation playing.

In this example, we will use two view windows. The camera object is modified through one view window, while another view shows how the scene looks through the camera.

- Open two view windows.
- Create a camera object with View/Camera/Define Camera.
- Set View/Camera/Camera View for the second view window.
- Modify the viewpoint or the aimpoint object through the first view window. The orientation of the second view is automatically updated while you move the mouse.

Render/Settings

Auto Box Rendering

The Render settings dialog contains an 'Auto Box Rendering' option. This switches on the new box rendering feature and tries to optimize the use of the processors internal cache memory. The higher the image resolution, the larger the time saving obtainable by this optimization. The speed improvement can be significant, especially on a PC, delivering up to twice the rendering speed. It should be noted that when rendering simple scenes to a disk file, the overhead of extra disk operations may actually slow rendering.

Interlace fields

The Render settings dialog contains a new control for field rendering. If the 'Interlace Fields' option is selected, Real 3D automatically interlaces every two fields into a frame which is saved, and then deletes the fields.

Super sampling

This is another new render setting. When this option is selected, the X- and Y-resolution fields define sampling density on sub-pixel level. Instead of rendering several pixels by one sample, which is the normal function of the X- and Y-resolution fields, several samples are taken from one pixel when Super sampling is selected.

For example, if super sampling is set and X- and Y-resolution are both two, the result is the same as if you rendered a 4 times higher resolution image and then scaled it afterwards by averaging each 4 pixel group into one pixel.

This feature can greatly improve image quality when the scene contains small details or dense texture maps. The disadvantage is slower rendering time: it takes almost four times longer to render an image with 2*2 super sampling.

Note: if you use super sampling to improve image quality, you may safely lower the normal anti-aliasing level. Typically anti-aliasing factor two is enough when super sampling is applied.

Alpha channel output in IFF file rendering

Iff rendering to a file with Alpha output selected now produces an 8 bit alpha file instead of 1 bit mask file.

JPEG support

It is now possible to render JPEG format images.

Saving and loading render settings

Render settings can be named, saved and reloaded in a similar way as grids, fractal settings and materials. You can define a memory resident collection of individual, named render settings, which may be saved to disk as a render settings library. Whenever you need good render settings, you can reload this kind of library and select a suitable entry from it instead of redefining the settings from scratch.

The render settings dialog contains a 'Name' edit box where you can define a name for the currently edited render settings. Pressing the Save button adds a new entry to the settings library. You can use the Load control to load previously defined settings back to the dialog. The Delete button is to remove entries from the library.

A render settings library can be saved to disk with the File/Project/ Save sections function if you check only the Render settings section. Settings can be reloaded with File/Project/Insert, Replace, Insert sections or Replace sections functions.

Pressing the OK button in the render settings dialog doesn't update previously loaded render settings. Use the Save button to write modified settings to the settings library.

Note: it is usually not necessary to define render setting libraries. Each view window includes its own render settings and when you save a project, the settings are saved with view windows. Render setting libraries are necessary when you intend to re-use carefully defined settings from other projects or want to copy settings from one view window to another one.

Post effects

Real 3D's post effect system makes it possible to build a list from 'post effect modules' which each manipulate the rendered image. This way, a variety of easily controllable, interesting effects can be added to images quickly.

Some post effects are computed automatically by Real 3D. You can add extra 'plug in' post effects to the program. This happens by placing a library which defines such effects into the 'Classes' directory in the Real 3D home directory. This way, the post effect library is automatically installed whenever Real 3D is started. Of course, the 'Classes' directory may contain several post effect libraries. It is also possible to open a new library without auto installation at program startup time with the File/Windows/External Classes window. This window can be used to select and open extension libraries for Real 3D.

The render setting dialog contains a 'POST EFFECTS' button. This button opens a dialog where the post effect list can be constructed and edited. Each view window can have its own post effect list.

The post effect dialog



The Post effect dialog has two list boxes. The left one displays all installed post effect libraries. You can select a library from this list. If you then press the 'Create' button at the bottom left corner of the window, a new effect is created and inserted to the view window's post effect list. You may then edit its properties by pressing the 'Edit' button. This opens a new dialog, whose contents depends on the effect in question. See the effect library specific documentation for details.

The list selector at the right side of the window displays the post effect list created so far. You can activate a list item and edit it, or delete it with the 'Delete' button. The image which the renderer computes is processed by the effects in this list. The topmost effect receives the image data first, manipulates it, sends it to the next effect and so on. This means the order in which the effects are computed may have some importance. When you create a new effect, it is inserted after the currently selected effect. If none of the effects are selected (e.g. when the post effect dialog has been just opened), the newly created effect is inserted to the beginning of the list.

If a particular post effect has been inserted to the post effect list, but the corresponding effect library isn't opened when rendering starts, the effect will not be active. For example, this may happen if you send a project to another Real 3D user who doesn't have the same effect libraries installed on his/her system.

Glow

The glow effect processes glow size and brightness material properties.

Glow effect has two global attributes: an Additive option and a Fixed size option.

Additive:

When this button is set, the brightness of overlapping glow areas is combined to a higher brightness value. Additive glow usually creates a brighter glow effect, but sometimes strong glow brightness variance may be undesirable. Note also that additive glow makes the object to which it is applied brighter, so usually the color of the glowing object may be quite dark.

Fixed size:

When this option is set, the glow is computed as a 'camera effect', and glow size doesn't depend on glow source position, distance from camera or image scale. This kind of glow is slightly faster to compute than the default 'space volumetric' glow.

Distance anti-aliasing

Distance blur is a simple post effect, which can be used to improve anti-aliasing of distant objects. The effect smoothens the image output depending on the distance from the camera.

This effect has one attribute, 'Distance blur'. It defines the blurring sensitivity. The higher the value, the closer objects become blurred. A good default value is 30.

Render/Export DXF

This function exports polygonal objects and freeforms as surfaces in DXF format. Quadric primitives are exported as curve data corresponding to the wireframe representation.

Drawing Settings

The new, fast backdrop image blitting feature is enabled by the "Blit Backdrop" checkbox in the Drawing Settings dialog.

To use this feature, use the Render Settings dialog to define the Backdrop image file name. Set the "Blit Backdrop" checkbox in the Drawing Settings dialog. After that, Real 3D will draw the defined image in the view window before wireframe refreshes. The image is rendered in a similar way as the backdrop image is in ray trace rendering. In other words it is rescaled to match the dimensions of the view window.

This function can handle all kinds of images supported by Real 3D. If the display mode doesn't correspond to the image, conversion is applied.

The purpose of this feature is to allow accurate perspective matching, rotoscoping, etc.

Animate/**Create/Keyframe**

Builds up a hierarchy required for keyframe animations:

- Creates a new level, and puts the selected object into the level.
- Creates a keyframe method
- Defines the first key to represent the current state of the selected object for the keyframe method.

Select the object to be animated before you activate this function. After you apply the function, you can continue by making the created keyframe method the current level and use the key editor (Animate/Edit) to define more key frames.

See the tutorials, Animate/Edit and chapter 3 'Animation system' for more information.

Create/Inverse Kinematic

Select a skeleton and then this function. Select suitable path options and draw a motion path for a joint. The function creates a new level, copies the skeleton into it and creates an inversed kinematics method with the defined parameter curve. The defined path is automatically associated with the nearest joint.

If you need to define more than one inversed kinematic curve, don't apply this function again to the skeleton, but make the newly created inversed kinematic method the current level and create more curves inside the method level.

Create/Skeleton

Builds a hierarchy where the selected objects are controlled by a simple skeleton method.

To use this function:

- Select the objects you want to control with a simple skeleton method.
- Pick the menu Animate/Create/Skeleton.
- Draw a skeleton primitive in a view window.
- Define suitable skeleton attributes in the opened skeleton dialog.

This function automatically maps targets which are freeforms onto the skeleton as one point sub groups. The purpose of this is to allow 'skin' like control of freeforms (usually some fidelity needs to be added to the points near the joints). If you want to control a freeform as a one single block, put it under a level object before you apply this function to the level.

The function doesn't immediately fix the targets to the skeleton. The targets become attached the first time the animation is played/refreshed. The reason the function operates in this way is to allow editing, fine adjustments of the defined skeleton primitive and definition of possible sub skeletons before the targets become attached.

Create/Shrink wrapping

This menu function creates an animated shrink wrapping effect. Use the function in the following way:

- Select the object, against which you want to shrink another object.
- Multi-select (keep the shrink target selected!) the object to be shrunk, for example a freeform mesh.
- Pick this function from the menu.
- A dialog is opened. Select the shrink wrapping type. If you use parallel shrink, you may also select the cumulative option. Click OK.
- Depending on the shrink wrapping type, Real 3D waits you to define some parameter information. If you selected cylindrical or spherical shrink wrapping, define first the shrinking center by a mouse click and then draw a line defining the amount of shrinking. If you selected normal or parallel shrink wrapping, just draw a line defining the amount of shrinking (and the direction in parallel wrapping).

Edit

You no longer have to select a animation method prior to using the function Animate/Edit. The function still attempts to open a method sensitive editing window, but if selected objects don't contain any animation methods, the key editor is opened instead.

For more information about the key editor, see chapter 3 of this appendix - 'Animation system.'

Extras/**Vectors/Deselect Group**

This function removes any points from the vector stack which are also in selected groups when the function is used. The purpose of this function is to make it easier to subdivide complex freeform objects into sub groups so all points of the freeform object in question belong to one, and only one group.

Example: let's subdivide a B-Spline object into two separate groups:

- Load or create a b-spline object
- Shift-drag or lasso select some points of the B-Spline object and create a group with the function Create/Structure/Group.
- Now select all points from the b-spline object, for example by Shift-dragging. Make sure the previously created group is selected and then use the Deselect Group function.
- Select Create/Structure/Group. The new group contains all points except the points of the first group.

Repeat Last

The function Extras/Repeat Last re-executes (re-activates) the previously executed function. The hot key is '5'.

Show selected objects -hotkey

Selected objects can be shown in a view window as flashing wireframes by pressing the '0' hotkey on the numeric keypad. This key binding is built-in.

Settings/

Attributes

The Settings/Attributes function can now be used to define default names and depths for all objects. Sometimes this may be extremely useful. For example, if you are designing kitchen cabinets, it might be a good idea to set the initial depth of the cube primitive to 15, because 15 mm. is the thickness from which the kitchen cabinets are most commonly manufactured.

The changed primitive attributes can be saved with a project, or to an environment file (for example to startup.r3d), with for example File/Project/Save sections (Primitive Attributes section should be checked).

General

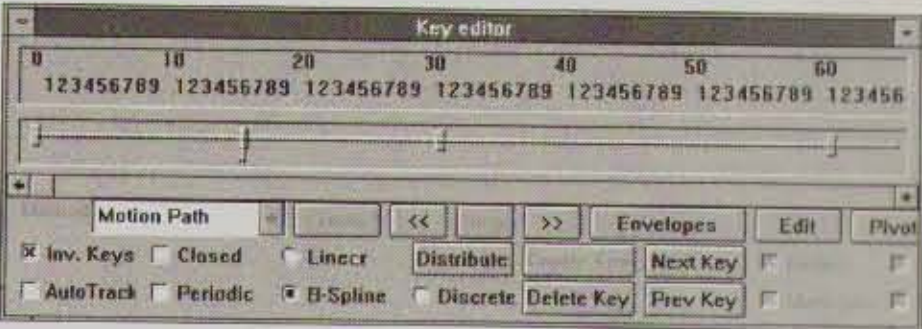
A "No RPL busy requester" checkbox has been added to Settings/General. If it is checked, no busy requester is opened when a named macro is executed, or when an RPL file is otherwise loaded.

3. ANIMATION SYSTEM

THE KEY EDITOR

Many of the new features have been integrated into the key editor window, which is opened with the menu Animate/Edit.

The key editor



The window consists of the following fields:

- Frames:**
It is now possible to move in time by clicking the desired frame number with the left mouse button.
- Knot sequence:**
This is the raised bevelled box just below the 'frames' field. It shows the knot sequence of the method currently being edited. In v.2, this was only used to control the morphing method. Now handles all path oriented methods and the new keyframe method.
- Invisible:**
This check box makes the method and any other objects within the method hierarchy either Wireframe Visible or Wireframe Invisible.
- Auto Track:**
This check box enables Auto scrolling within the key editor window during animation playback.

Close:

Closes the evaluation of a motion. For path animations, this is equivalent to applying the Modify/Freeform/Open/Close function to the parameter curves.

Periodic:

Defines periodic evaluation, which makes closed motions smoothly repeatable. See Animate/Create/Path/Periodic.

Linear/B-Spline/Discrete:

This defines the type of interpolation used.

Method:

This cycle gadget defines the method to be created. If the method is 'Transform', a time transformation is created and inserted at the beginning of the current level.

Create:

Creates a method defined by the 'Method' cycle gadget. The function is identical to items found in the Animate/Create menu.

Envelopes:

Opens the envelope editor. See the description later in this section.

Distribute:

Redistributes the knot sequence uniformly over the time and causes target objects to spend equal time between all subsequent keys/knots. This also handles closed motions appropriately.

Create Key:

This creates a new key object/knot point. If the method is keyframe or morphing, a new key object is created. If the method is one of the path oriented methods, a new knot point is created for the parameter curve.

Note: this function can only be applied when there is no key object/ knot point which corresponds to the current time.

Delete Key:

This deletes the current key object/knot point. This function can be applied only when the current time matches an existing key object/knot point.

Next:

Plays the animation to the next key-object/knot point. If the animation is played to the end, this button is disabled.

Prev:

Finds the previous key object/knot point.

<< STOP >>:

These buttons can be used to play the animation backwards and forwards, and to stop playback at any point.

Edit:

When editing keyframe or morphing methods, the 'Edit' button prepares the system for key editing. If the current time matches an existing key, this button selects the key so the modifications are performed for the correct object. If the time doesn't match any key, no keys will be selected and only the target object will become selected.

Pivot:

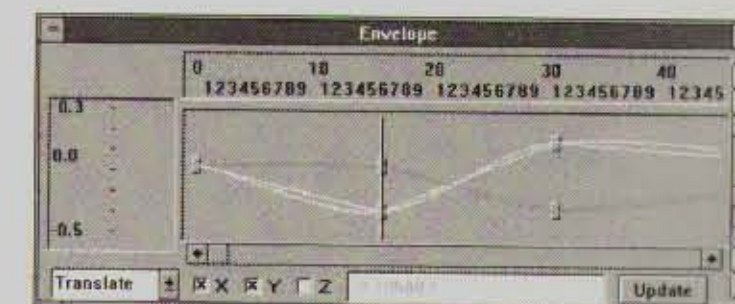
Selects the first object in the current level, and calls the Modify/Properties/ COG function, so you can redefine the COG with a left mouse button click. This is convenient when creating keyframe animations. The keyframe method rotates the target object around the COG (pivot) point.

Whenever full rotation control is needed, it is best to define the pivot point, because the default pivot may not be suitable. The pivot point should be defined before the keys have been created, because otherwise, redefinition of the pivot may change the animation quite radically, requiring a lot of re-editing. The pivot point definition may be done conveniently with this button immediately after a keyframe method has been created with the Create button of the key editor.

To edit an animation method with the key editor, make the method the current level.

ENVELOPES

The envelope editor is opened by clicking the 'Envelope' button in the key editor window. Dimensions (x, y, z) can be switched on separately with X, Y and Z check boxes. The envelope editor shows envelopes of the current level, provided it is a path oriented or a keyframe method.



The envelope editor

+/- buttons on the right corners are used to change the minimum/maximum limits the envelope editor is displaying.

To modify an envelope, just click the desired 'knot point' and move it to a new position.

Note: you can't modify knots in the horizontal (time) direction. Use the key editor for this purpose.

While editing knot points, you can enter the position numerically in the 'Value' string gadget.

It's also possible to move more than one knot by dragging.

The 'Update' button updates the animation system so you can see the effect of the envelope modification in the actual scene.

Four envelopes are available for the keyframe method:

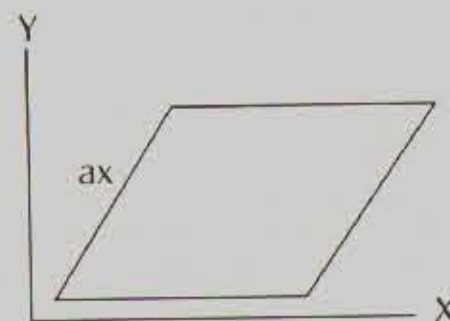
- Translate (move)
- Scale
- Rotate
- Shear (skew).

Translate shows the relative motion in object's space, i.e. how much the object is displaced from its original position.

Scale defines how much the size of the object is changed. A value of 1 means no change. A value of -1 means object is mirrored. A value of 2 means the size is doubled.

Rotate is expressed in degrees and is relative (it shows how much the object is rotated from its initial state).

The following figure shows how shear is defined.



Shear is defined by three angles:

- ax - defines how much the 'y' component of the object is skewed towards the 'x' axis.
- ay - y towards the 'z'
- az - z towards the 'x'

The envelope window may also be used to modify paths of all path oriented methods such as:

- sweep
- size
- stretch
- inversed kinematics
- move
- direction

Only the translation envelopes of the abovementioned methods can be edited.

PATH ANIMATIONS

Polygonal and B-spline curves now contain so called 'knot sequence', which can be used to define how the time is distributed over the curve. In other words, you can define the frame in which the target object reaches each knot point of a curve.

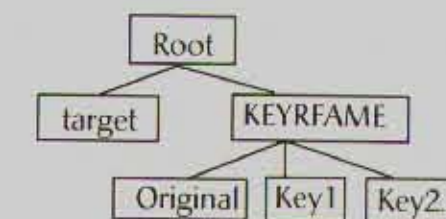
The key editor can be used to create path oriented animation methods and to edit knot sequences. A suitable animation method can be selected from the 'Method' list and the 'Create' button activates a method creation process, in which a curve is defined and drawn in a view window.

To edit a path method with the key editor:

- Open the key editor from the menu Animate/Edit. Make sure the path method isn't the current level, otherwise you will get a path edit dialog.
- Make the path method the current level.
- Next and Prev buttons of the key editor can be used to jump from one knot point to another.
- Create Key and Delete Key buttons can be used to create/delete knot points to/from motion paths.
- The timing can be adjusted by moving the knots along the time line.

KEYFRAME METHOD

The syntax for the keyframe method is as follows:



In other words, the shape of the 'target' is defined by applying a transformation to the 'original' object. The transformation is defined by interpolating a number of keyframe method specific tags associated with 'key1, key2, ...keyn' objects.

Four tags are used to store the required transformations to key objects:

- VMOV - This tag defines how much the target object is displaced from the 'original' object in x, y and z directions.
- VROT - This tag defines how much the target object is rotated in 'x', 'y' and 'z' directions.
- VSCA - This defines how much the target object is scaled in its 'x', 'y' and 'z' directions.
- VSHE - This defines how much the object is sheared (skew transformation).

IMIT tag defines whether to use linear, B-spline or discrete interpolation. The IFLG tag selects open/closed interpolation. See the morphing method description for more information.

The pivot point about which rotations are interpolated is defined by the COG of the original object i.e. the first parameter object of the keyframe method. The default COG value can be redefined by applying the Modify/Properties/COG function.

Note: the easiest way to create keyframe animations is to use the key editor, and normally you don't have to worry about the tags mentioned above.

To create a keyframe animation:

- Open the 'Key Editor' window by selecting the menu Animate/Edit. Set the 'Method' selector to 'Keyframe'.
- Select an object to be animated (or create it) and click the 'Create' button. This creates a keyframe system which consists of one key frame.
- To create more key frames, move to a desired position in time by clicking the desired frame, click the 'Create Key' button and modify the new key object.
- In order to modify other key frames, use the 'Next' and 'Prev' buttons to select the desired key. Note that these functions automatically select the target object and the correct key frame object, so you can instantly start editing the key.
- Only linear transformations (Modify/Linear menu) can be used to modify key objects. Functions such as 'Bend' and 'Twist' can't be used (use morphing, if you need non-linear transformations).
- If you want to edit a previously created key frame method, make it the current level and press the 'Edit' button.

MORPHING METHOD

The MORPHING_CLOSED method no longer exists. There is only one Morphing method available. Open morphing is used by default. Closed morphing can be selected by editing the method with the key editor (Animate/Edit), which contains a 'Close' button for this purpose.

Internally, the IFLG tag is used to define whether or not open evaluation should be used. If the first bit (value 1) is set, closed interpolation is used.

The morphing method recognizes a new interpolation type: discrete. The discrete interpolation changes the target object only at key frames. This could be used, for example, to create 'camera cut' effects. The tag IMIT defines which interpolation type is used. Possible values of the IMIT tag are:

- 0 - Linear interpolation
- 1 - Cubic B-Spline interpolation
- 2 - Discrete

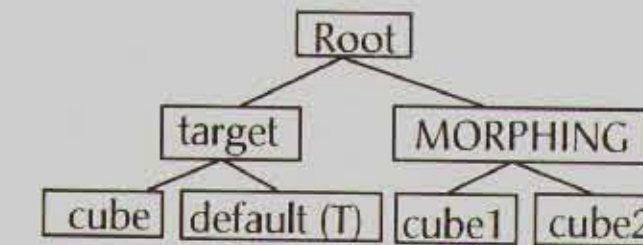
The interpolation type can be selected in the key editor.

Another improvement is that all tags are now morphed (light source brightness etc.). Furthermore, you can selectively morph geometry, color, tags and materials. This is controlled by the tag IFLG associated with the method object. Again, the key editor includes the necessary controls.

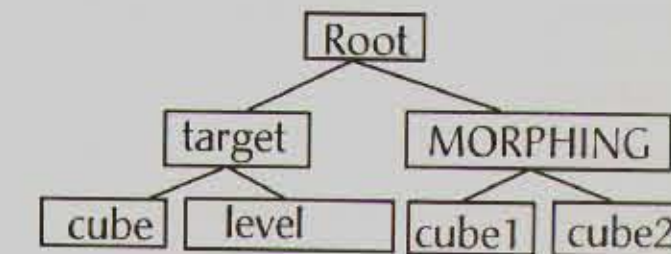
IFLG flag bits are:

- 0 - Set for closed, reset for open evaluation
- 2 - If set, materials are morphed
- 3 - If set, geometry is morphed
- 4 - If set, tags are morphed
- 5 - If set, object colors are morphed

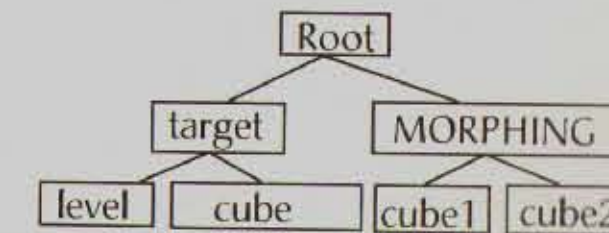
Depending on which data is selected by the abovementioned bits, the hierarchy of the target object and key objects need not match exactly. If geometry morphing is selected, then the primitives of keys and their hierarchical grouping must match the target object exactly, although there may be e.g. additional default mappings in the target object, because there is no geometry associated with that mapping type. For example, the following hierarchy is valid:



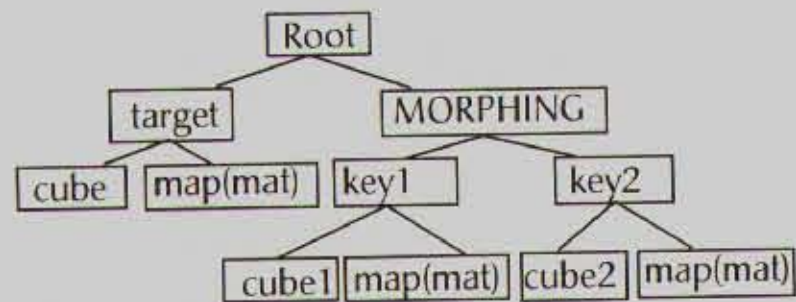
If color morphing is selected, the primitive hierarchy must match exactly, except there may be additional level objects at the end of hierarchy levels, because level objects don't contain color data. So this is again a valid hierarchy:



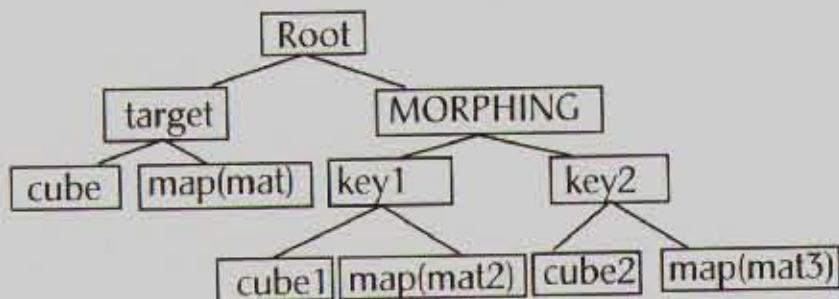
This doesn't work if either geometry or color morphing is selected:



Material morphing can usually be switched off, because if the keys are copies of the target object, all the mapping objects refer to the same materials in the material library, and morphing doesn't have any effect. So the only consequence of switching material morphing off is faster animation playback. For example, in the following hierarchy, all mapping objects 'map' refer to the same material 'mat':



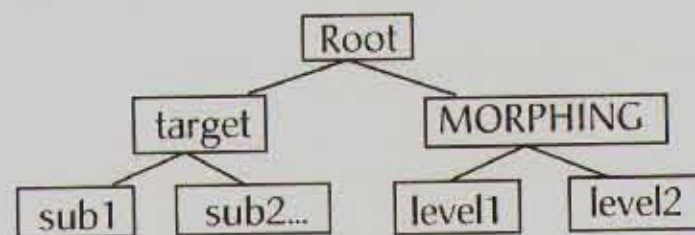
Morphing materials makes sense only if each of the materials is different, i.e. three separate materials mat1, mat2 and mat3 are used:



Tag morphing has no requirements for hierarchy matching. The animation system scans the hierarchy, picks objects according to their position in the hierarchy and morphs their tags. This process is continued until a key doesn't define the currently morphed target tag, i.e. a hierarchy difference is detected. A possible difference doesn't produce an error message, the rest of the tags of the target are just left unmodified.

Note: string tags (of the form Sxxx=...) can't be morphed.

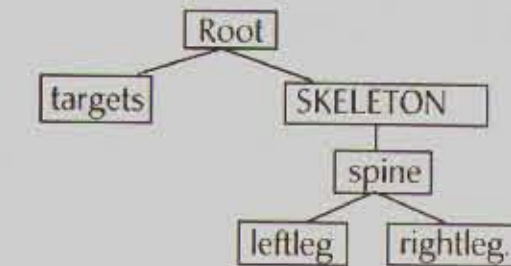
For example, tags of the root level of a compound object can easily be morphed as follows:



In this hierarchy, level objects 'target', 'level1' and 'level2' contain the tags to be morphed. Level1 and level2 can be empty levels. A useful application is morphing the fade attribute defined by IFAD tag this way.

SKELETON METHOD

The skeleton method handles hierarchical skeleton primitives. The actual syntax is exactly the same as in v.2.



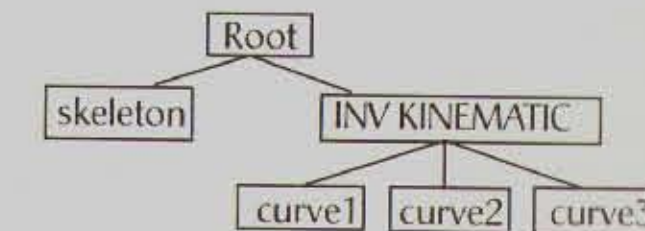
SIMPLE SKELETON METHOD

In this version, the simple skeleton method automatically maps target freeforms as one point sub groups onto the skeleton.

The parameter of a simple skeleton method may be a hierarchical skeleton.

INVERSE KINEMATICS METHOD

The inverse kinematics method can now take several parameter curves which are automatically attached to the nearest joints of the target skeletons.



The tag IIND, when associated with parameter curves of the inverse kinematics method, defines the joint to be affected by the curve in question. In addition to this, inverse kinematics also needs to know to which skeleton the joint index refers (thanks to the new hierarchical skeleton feature, the target skeleton may be hierarchical; in which case there are actually several skeletons). The SIDE tag is used for that purpose. For example, if 'curve1' should define a motion for the third joint of 'skel21' there should be the following tags associated with 'curve1':

IIND=2 (third joint, 0 is the last = default)

SIDE=xxx.x

and the tag 'SIDE=xxx.x' must also be attached to 'skel21'.

Note: these tags are defined automatically when the animation is played or refreshed for the first time. If you don't want automatic definition, add an ISKE=2 tag to the method before you play the animation. If you want the program to redefine these tags, delete the ISKE tag.

SURFACE METHOD

This new method is based on collision detection. The method makes an animated object automatically travel along a surface. The motion is generated by user defined 'friction points': the method doesn't let these points slide along the surface, but moves the target object instead.

The surface method takes three parameter objects:

1. Gravity:

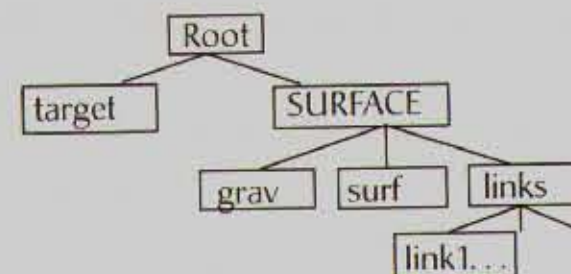
This defines the direction for gravity. For example, you can draw an axis (Create/Controls/Axis) to define the direction. Note that this parameter may be animated so the direction of gravity doesn't remain the same.

2. Surface:

This represents the ground. This could be any object, such as a mesh surface.

3. Links:

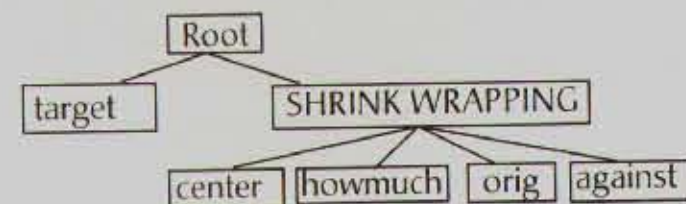
A level object which consists of a number of links. Links must refer to offset primitives in the target object. These offsets are taken as 'friction points'.



SHRINK WRAPPING METHOD

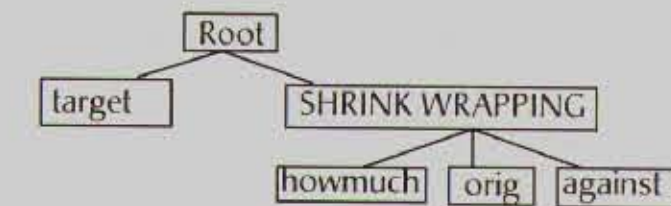
This method is the animated version of Modify/Special/Shrink wrap functions.

The syntax of cylindrical or spherical shrink wrapping is as follows:

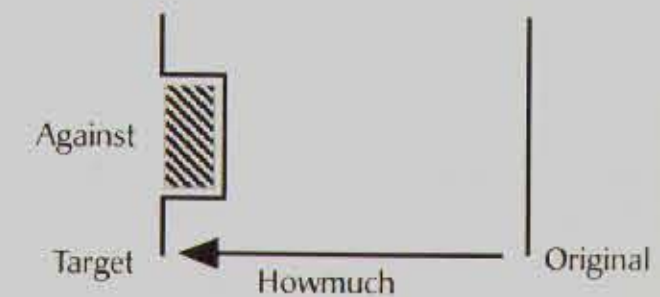


- center - This can be any evaluable object (such as an offset or a curve) and defines the center for shrink wrapping. This corresponds to the first mouse click in the interactive shrink wrapping functions.
- howmuch - This defines how much the object to be shrunk is 'scaled'. Normally, this should be an axis. The end points of the axis correspond to the second and third mouse clicks in the interactive shrink wrapping functions.
- orig - This defines the 'original shape' of the target object. The hierarchical structure must match the target object exactly.
- against - this corresponds to the first selected object in the interactive shrink wrapping. In other words, 'orig' object is shrunk around this object.

Parallel and normal shrink wrapping are animated with the following, slightly simpler hierarchy:



In parallel and normal shrinking there is no 'center' object. The 'howmuch' curve just defines how much and in which direction the original object is moved.



There are two possible ways of shrinking: cumulative and non-cumulative. In cumulative shrink wrapping, the deformation caused by the shrink wrapping is assigned back to the original object. In non-cumulative shrink wrapping, the shape of the original object isn't affected and remains the same.

The tag ITRA is used to select the type of shrink wrapping:

- 50 - parallel
- 54 - cylinder
- 55 - sphere
- 45 - normal

If the tag isn't defined, parallel is used as a default.

The tag IFLG selects cumulative shrink wrapping:

- 0 - non-cumulative (default)
- 1 - cumulative

These tags are attached to the shrink wrapping method object.

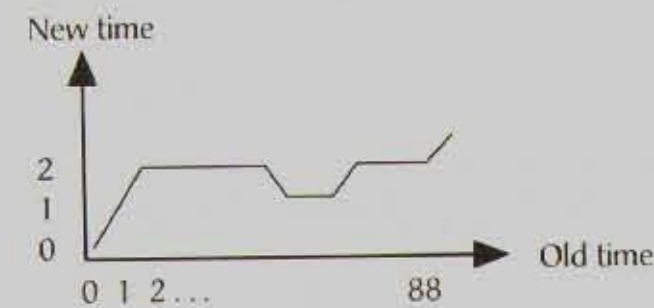
TRANSFORM METHOD

A Transform method can now be easily created from within the key editor. Just set the 'Method' selector to 'Transform' and click the 'Create' button. This creates a transform method with the necessary parameters: a coordsys and a b-spline curve. Then the envelope editor can be used to modify the default curve.

Note: the created transform method is inserted ahead of all the existing objects in the current hierarchy level, so the defined time transformation affects all objects in the current level.

There are currently two envelopes available for this method:

- Translate This shows the curve components in absolute space.
- Time This shows how the time is mapped to a new time. The old time is represented by the 'x' axis and the new time is represented by the 'y' axis.



Example:

Let's imagine the situation where a sphere moves along a straight line. Required accelerations/decelerations are created by a transform method.

- Open the key editor by selecting the function Animate/Edit.
- Create a sphere and make it follow a path by setting the method in the key editor window to Path and click the Create button. Select 'Polygonal' from the path dialog and draw a straight, two point line as the path.
- Create a transform method by setting the cycle gadget in the key editor window to Transform and then click the create button.
- Open the envelope editor by clicking the Envelope button in the key editor window.
- Set the Envelope selector to 'Time' and modify the 'X' curve.

4. RPL

LOCAL VARIABLES

RPL now supports local variables (automatic variables). A variable is 'local' if it is defined inside a word definition.

```
: test
  VARIABLE tmp1
  VARIABLE tmp2
```

```
  10 tmp1 STORE
```

```
  ...
```

```
;
```

A local variable is visible and defined only inside the word in question.

PARAMETERS

Another new feature introduced by V3 is 'parameters'. Parameters are listed at the beginning of a word definition:

For example, the following word:

```
: TEST
  PARAM
    VARIABLE iAge
    VARIABLE aName
    FVARIABLE fMass
  ENDPARAM
```

```
  ...
```

```
;
```

can be called as

```
21 "Joanna" 65.5 TEST
```


NEW RPL WORDS

V3 contains a small but powerful set of words for manipulating windows, buttons and other display elements. In order to use these words, the file `rpl/sys/ui.rpl` must be loaded first.

Each word takes certain number of fixed parameters plus some optional tag-value pairs. A tag list must be terminated by `UI_Done`.

Also many other new RPL words have been included.

ADDMENU	
SYNTAX	aWord sName ADDMENU
DESCRIPTION	Adds a new menu item to the menu strip. When the menu is selected, the associated RPL word is called. If the name string contains the character <code>'/'</code> , a new sub item is created. Otherwise a new menu item is created.
EXAMPLE	This example creates a new menu item 'MyTools' consisting of two sub items: 'Fancy Tool' and 'Cool Tool'. : myFancyTool "Fancy Tool selected" PUTS ; : myCoolTool "Cool Tool selected" PUTS ; & myFancyTool "MyTools/Fancy Tool" ADDMENU & myCoolTool "MyTools/Cool Tool" ADDMENU

COMPARE	
SYNTAX	sFirst sSecond COMPARE iCmp
RETURNS	The return value indicates the lexicographic relation of sFirst to sSecond, as follows: iCmp < 0 sFirst is less than sSecond iCmp = 0 sFirst is equal to sSecond iCmp > 0 sFirst is greater than sSecond
PARAMETERS	sFirst, sSecond - strings to be compared
DESCRIPTION	Compares two strings lexicographically and returns a value indicating their relationship.

CS_DEFINE	
SYNTAX	vX vY vZ vOrigin CS_DEFINE
PARAMETERS	vX vY vZ - Three axes defining the orientation of the coordinate system vOrigin - Origin
DESCRIPTION	Defines the current coordinate system for CS_ROTATE, CS_SCALE, CS_TRANSLATE and CS_TRANSFABS words.
EXAMPLE	0 0 0 1 0 0 0 1 0 0 0 1 CS_DEFINE

CS_ROTATE	
SYNTAX	0 ... aObj vDir fAngle CS_ROTATE
DESCRIPTION	Rotates objects about a given axis that passes through the coordinate origin. Rotation is applied in the current coordinate system.

O_GETSEL	
EXAMPLE	1 0 0 90 CS_ROTATE
SEE ALSO	CS_DEFINE

CS_SCALE	
SYNTAX	0 ... aObj vScale CS_SCALE
DESCRIPTION	Scales the given objects in the current coordinate system.
EXAMPLE	O_GETSEL 0.5 1 1 CS_SCALE
SEE ALSO	CS_DEFINE

CS_TRANSFABS	
SYNTAX	0 ... aObj CS_TRANSFABS
DESCRIPTION	Modifies the given objects so that their object spaces will match the current coordinate system.
SEE ALSO	CS_DEFINE

CS_TRANSLATE

SYNTAX	0 ... aObj vMove CS_TRANSLATE
DESCRIPTION	Moves the given objects. The transformation is applied to objects in the current coordinate system.
EXAMPLE	0.1 0 0 CS_TRANSLATE
SEE ALSO	CS_DEFINE

DATETIME

SYNTAX	DATETIME iSec iMin iHour iMday iMon iYear
RETURNS	Current date and time
DESCRIPTION	Returns the current date and time as follows: iSec - seconds after the minute (0-59) iMin - minutes after the hour (0-59) iHour - hours since midnight (0-23) iMday - day of month (1-31) iMon - month (1-12, january = 1) iYear - current year

FCLOSE

SYNTAX	aFile FCLOSE iSuccess
RETURNS	0 if FCLOSE succeeds
PARAMETERS	aFile - pointer to a file
DESCRIPTION	Closes the file opened by FOPEN.

FEOF

SYNTAX	aFile FEOF iAtEof
RETURNS	nonzero if at end of file, 0 otherwise
PARAMETERS	aFile - pointer to a file
DESCRIPTION	Returns a nonzero value after the first read operation that attempts to read past the end of the file. It returns 0 if the current position is not end-of-file.

FGETS

SYNTAX	sString iMaxLen aFile FGETS iSuccess
RETURNS	sString if FGETS succeeds, NULL otherwise

PARAMETERS	sString - buffer for data to be read
	iMaxLen - maximim number of characters to read
	aFile - pointer to a file

DESCRIPTION FGETS reads a string from the file to the buffer. Characters are read from the current file position up to and including the first newline character, up to the end of the file, or until the number of characters read is equal to iMaxLen-1, whichever comes first. The result is stored in sString, and a null character is appended. The newline character, if read, is included in the string.

FOPEN

SYNTAX	sFilename sMode FOPEN aFile
RETURNS	Pointer to the opened file or NULL if failed
PARAMETERS	sFilename - Filename sMode - Access type

DESCRIPTION Opens the file specified by filename.
The sMode parameter specifies the type of access requested for the file:

- "r" Opens for reading only. If the file does not exist or cannot be found, the FOPEN word will return NULL.
- "w" Opens a file for writing. If the given file exists, its contents are destroyed.
- "a" Opens for writing at the end of the file (appending); creates the file first if it doesn't exist.
- "r+" Opens for both reading and writing. The file must exist.
- "w+" Opens a file for both reading and writing. If the given file exists, its contents are destroyed.
- "a+" Opens a file for reading and appending; creates the file if it doesn't exist.

When the "r+", "w+", or "a+" access type is specified, both reading and writing are allowed. When switching between reading and writing, there must be an intervening FSEEK operation. The following characters can be included in sMode to specify the translation mode for newline characters:

t
Open the file in text mode. In text mode, carriage-return-linefeed (CR-LF) combinations are translated into single linefeeds (LF) on input and LF characters are translated to CR-LF combinations on output. Also, CTRL+Z is interpreted as an end-of-file character on input. In files opened for reading/writing, FOPEN checks for a CTRL+Z at the end of the file and removes it, if possible. This is done because using the FSEEK and FTELL words to move within a file that ends with a CTRL+Z may cause FSEEK to behave erratically near the end of the file.

b
Open the file in binary mode. The translations are not carried out.

FPUTS

SYNTAX	sString aFile FPUTS iSuccess
RETURNS	a non-negative value if FPUTS succeeds
PARAMETERS	sString - string to be written aFile - pointer to a file
DESCRIPTION	Writes the string to the file.

FREAD

SYNTAX	sBuffer iSize iCount aFile FREAD iRead
RETURNS	number of items fully read
PARAMETERS	sBuffer - buffer for data to be read iSize - size of one item iCount - number of items to be read aFile - pointer to a file
DESCRIPTION	Reads up to iCount items from aFile file to sBuffer. Each item is of length iSize.

FSEEK

SYNTAX	aFile iOffset iOrigin FSEEK iSuccess
RETURNS	0 if FSEEK succeeds
PARAMETERS	aFile - pointer to a file iOffset - number of bytes from origin iOrigin - initial position
DESCRIPTION	Repositions the pointer in a file. The iOrigin parameter can be any of the following: SEEK_CUR - current file position SEEK_END - end of file SEEK_SET - beginning of file

FTELL

SYNTAX	aFile FTELL iPosition
RETURNS	current file position
PARAMETERS	aFile - pointer to a file
DESCRIPTION	Returns the current file position.

FWRITE

SYNTAX	sBuffer iSize iCount aFile FWRITE iWritten
RETURNS	number of items written
PARAMETERS	sBuffer - data to be written iSize - size of one item iCount - number of items to be written aFile - pointer to a file
DESCRIPTION	Writes up to iCount items from sBuffer to aFile file. Each item is of length iSize.

LEN

SYNTAX	sString LEN iLen
RETURNS	The length of the string.
PARAMETERS	sString - string whose length is to be measured
DESCRIPTION	Returns the length of a string.

NCPY

SYNTAX	sDest sSource iCount NCPY
PARAMETERS	sDest - destination string sSource - source string iCount - maximum number of characters to copy
DESCRIPTION	Copies the first iCount characters of sSource to sDest. If iCount is less than or equal to the length of sSource, a null character is not appended to the copied string. If iCount is greater than the length of sSource, the destination string is padded with null characters.

NCAT

SYNTAX	sDest sSource iCount NCAT
PARAMETERS	sDest - destination string sSource - source string iCount - maximum number of characters to append
DESCRIPTION	Appends, at most, the first iCount characters of sSource to sDest. The initial character of sSource overwrites the terminating null character of sDest. If a null character appears in sSource before iCount characters are appended, NCAT appends all characters from sSource up to the terminating null

character. If iCount is greater than the length of sSource, the length of sSource is used in place of iCount. The resulting string is terminated with a null character.

NOISE

SYNTAX	vPos fMin fMax fPower iOctaves NOISE vResult
PARAMETERS	vPos - A parameter value for the noise function fMin, fMax - Clipping interval fPower - Defines variation speed. MUST be greater than 0. iOctaves - Level of detail of the noise. Must be greater than 0.
RETURNS	vResult - A noise vector
DESCRIPTION	A multi-purpose three dimensional noise function, which computes continuously and randomly varying vector value from the parameter vector. The parameter vPos can be any point in space. The result vector's x,y and z coordinate component values are always inside the range [0,1]. The 'Octaves' parameter defines the level of detail i.e. how many iterations of fractal noise is included in the result. The fMin, fMax and fPower parameters define an output value filter for the result. The initial coordinate values of the computed noise are clipped to the interval defined by fMin and fMax and then rescaled back to the [0,1] interval. In practise this means that the smaller the difference between fMin and fMax, the larger constant value areas are created. Finally, the noise result components are raised to the power 'fPower'. The bigger the value, the more rapidly noise values change from one extreme to the other. Good default values are for example: - Octaves = 3 - fMin = 0, fMax = 1 (no clipping) or fMin = 0.3, fMax = 0.7 (some clipping) - fPower = 1 (Linear variation speed) or fPower = 4 (rapid variations)

O_MAKENAME

SYNTAX	aObj sName O_MAKENAME
DESCRIPTION	Builds up a string consisting of a full object path name from the given object.
EXAMPLE	(print the name of the first selected object 400 STRING sName O_GETSEL sName O_MAKENAME sName PUTS
SEE ALSO	O_FIND

PRJ_SETHOOK

TEMPLATE	aHook PRJ_SETHOOK
PARAMETERS	aHook - address of the RPL hook word
DESCRIPTION	This words allows you to read and process input plane 3d coordinates directly. The word sets a callback hook for all view windows. The hook procedure gets notification messages when the mouse is moved over any active view window or when a mouse button is pressed or released over a view window. The user must remove the hook when no longer needed. To remove a callback hook, call PRJ_SETHOOK with 0 as the callback address, i.e. 0 PRJ_SETHOOK
EXAMPLE	<pre>"ui.rpl" LOAD 100 STRING sBuf : cbViewWindow PARAM FVARIABLE PosX FVARIABLE PosY FVARIABLE PosZ VARIABLE iEvent ENDPARAM UIWM_Move iEvent FETCH = IF PosX FFETCH PosY FFETCH PosZ FFETCH "View mouse pos %log %log %log" sBuf SPRINTF sBuf PUTS ENDIF UIWM_LMBDown iEvent FETCH = IF "View mouse button clicked" sBuf CPY sBuf PUTS ENDIF UIWM_LMBUp iEvent FETCH = IF "View mouse button released" sBuf CPY sBuf PUTS ENDIF ; (set callback hook for view windows & cbViewWindow PRJ_SETHOOK</pre>

REMOVE

SYNTAX	sFileName REMOVE iSuccess
--------	---------------------------

RETURNS 0 if REMOVE succeeds, -1 otherwise

PARAMETERS sFileName - name of the file to be removed

DESCRIPTION Deletes the file specified by sFileName

SCANDIR

SYNTAX sPaths sCallBack aData SCANDIR

PARAMETERS sPaths - paths to be scanned

sCallBack - name of the RPL word to be called for each file found

aData - any user specified data to be forwarded to the RPL callback

DESCRIPTION Scans the directories given in the sPaths parameter. For every file found the sCallBack RPL word is called. The directories in the sPaths parameter are separated by a semicolon (;). The callback function should leave a positive value on the stack to continue scanning. A value of 0 or a negative value terminates scanning. The callback function has two parameters, full path name of the file and the user specified data.

EXAMPLE

```
VARIABLE count
: MyCallBack
  DUP FETCH 1 + SWAP STORE ( add 1 to count )
  "File is " PUTS PUTS 10 13 EMIT EMIT
  1 ( go on scanning )
;

"macro;rpl" "MyCallBack" count SCANDIR
count FETCH .
```

UI_WINDOW

SYNTAX ... aCallBack iLeft iTop iWidth iHeight sTitle UI_WINDOW aHnd

RETURNS Handle to the opened window or NULL if failed

FIXED PARAMETERS

aCallBack - Callback word for the window to be created

iLeft, iTop - Left top edge for the window

iWidth, iHeight - Size of the window

sTitle - String for the window title bar

TAGS None

DESCRIPTION Opens a window. Currently only the mouse events are reported to the callback. The callback takes the following parameters:

```
: cbWindow
  PARAM
    VARIABLE iEvent
    VARIABLE iMouseX
    VARIABLE iMouseY
  ENDPARAM
;
```

Callbacks must not return any values.

The iEvent parameter can be one of the following:

UIWM_Move (Mouse moved

UIWM_LMBDown (Left mouse button clicked

UIWM_LMBUp (Left mouse button released

iMouseX and iMouseY reflect the current position of the mouse.

UI_BUTTON

SYNTAX ... aWindow aCallback aLeft aTop aWidth aHeight sText UI_BUTTON aHnd

RETURNS Pointer to the created button or NULL.

FIXED PARAMETERS

aWindow - Pointer to the window created by using UI_WINDOW

aCallback - Word which is called when the user clicks the button

iLeft, iTop - Left-top edge of the button in a given window

iWidth, iHeight - Size of the button.

sText - Text for the button.

TAGS UI_Disabled - 1 = disabled, 0 = enabled

DESCRIPTION Creates a button in the given window. When the user clicks the button, the word pointed by aCallback is called. The callback doesn't take any parameters

```
: cButton
  "Button Clicked" PUTS
;
```

UI_CHECKBOX

SYNTAX ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_CHECKBOX aHnd

RETURNS Handle to the created checkbox

FIXED PARAMETERS

See UI_BUTTON

TAGS UI_Disabled - 1 = disabled, 0 = enabled
 UICB_Checked - 1 or 0 depending on the desired state of the check box.

DESCRIPTION Creates a checkbox. The default state is 'not checked'.
 The callback is called with one parameter which defines whether or not the checkbox was set or reset:

```

: cbCheckBox
  IF
    "Checked" PUTS
  ELSE
    "Not checked" PUTS
  ENDIF
  ;
  
```

UI_STRING

SYNTAX ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_STRING aHnd

RETURNS Address of the created string gadget or NULL if failed

FIXED PARAMETERS

See UI_BUTTON

TAGS UI_Disabled - 1 = disabled, 0 = enabled
 UIST_String - String for the edit control

DESCRIPTION Creates a string gadget (edit control). When the callback word is called depends on the platform. In Amiga, the callback occurs when the user pressed either Enter or Tab in the string gadget. In Windows, the callback occurs when the user leaves the edit control (either by using the Tab key or by activating another control by using the mouse).

```

: cbString
  "The user entered the string: %s" PUTS
  ;
  
```

UI_TEXT

SYNTAX ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_TEXT aHnd

RETURNS Pointer to the created gadget

FIXED PARAMETERS

See UI_BUTTON.

TAGS UITX_Text - Text to be displayed
 UITX_Border - If 1, the text will be surrounded by a frame

DESCRIPTION Creates a 'read-only' text gadget.

NOTE Because this creates a 'read-only' gadget, the parameter 'aCallback' is unused and can be NULL. Also the general purpose tag UI_Disabled has no effect.

UI_SLIDER

SYNTAX ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_SLIDER aHnd

RETURNS Pointer to the created slider

FIXED PARAMETERS

See UI_BUTTON

TAGS UI_Disabled - 1 = Disabled 0 on enabled
 UISL_Min - Minimum value
 UISL_Max - Maximum value
 UISL_Level - Current value

DESCRIPTION Creates a slider. When the user plays with the slider, the callback is called with one parameter reflecting the current level of the slider.

```

: cbSlider
  "Current level=%d" PUTS
  ;
  
```

UI_MX

SYNTAX ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_MX aHnd

RETURNS Address of the created gadget

FIXED PARAMETERS

See UI_BUTTON

TAGS UIMX_Active - Currently active item
 UIMX_Labels - Pointer array of strings terminated with null. For more information about how to create string pointer arrays using RPL, see rpl/ui.rpl.

DESCRIPTION Creates a 'Mutual Exclusive' gadget. The gadget consists of two or more choices and the user can select only one item at a time. In Amiga, the Cycle gadget is used. The Windows version uses 'Combo Box'.

The callback is called with one parameter which defines the ordinal number of the selected object.

```

: cbMx
  "You selected the item %d" PUTS
  ;
  
```


UI_DELETE	
SYNTAX	aHnd UI_DELETE
RETURNS	None
FIXED PARAMETERS	
	aHnd - Address of the window to be deleted (closed)
TAGS	None
DESCRIPTION	Deletes the given window. All the gadgets created on the window will automatically be deleted as well.

UI_SETATTRS	
SYNTAX	... aHnd UI_SETATTRS
RETURNS	None
FIXED PARAMETERS	
	aHnd - Handle to the gadget to be manipulated
TAGS	Possible tags depend on the type of the gadget.
DESCRIPTION	This word is used for setting window and gadget attributes. The parameter list consist of a number of tag-value pairs defining attributes and new values for them. For example, to change the current level of the slider to 10: <pre>UI_Done 20 UISL_Level aMySlider FETCH UI_SETATTRS</pre>

UI_GETATTRS	
SYNTAX	... aHnd UI_GETATTRS
RETURNS	None
FIXED PARAMETERS	
	aHnd - Handle to the gadget to be read
TAGS	Tags depend on the gadget in question
DESCRIPTION	This word can be used for reading attributes from the display elements. For example, to read the current value of the slider: VARIABLE aStrPtr <pre>UI_Done aStrPtr UIST_String UI_GETATTRS aStrPtr FETCH "The user typed: %s" PUTS</pre>

UI_REALIZE	
SYNTAX	aHnd UI_REALIZE
RETURNS	None
FIXED PARAMETERS	
	aHnd - Handle to a window to be realized
TAGS	None
DESCRIPTION	Displays all the created gadgets in the given window. The following example shows the basic structure of all GUI oriented programs: <pre>(create a window UI_Done ... UI_WINDOW aWnd STORE (create number of gadgets UI_Done ... aWnd FETCH UI_BUTTON aButton STORE UI_Done ... aWnd FETCH UI_... ... (Now the window is complete and must be realized aWnd FETCH UI_REALIZE</pre> A window needs to be realized only once.

CHANGES TO RPL SPECIFICATION

[&]	
	[&] word no longer exists. The word & can now be used everywhere where address of a word needs to be fetched.

SPRINTF	
	RPL word SPRINTF takes the parameters in a more natural order. e.g. <pre>40 STRING temp 1 2 3 "one %d, two %d, three %d" temp SPRINTF temp PUTS</pre> now prints <pre>one 1, two 2, three 3</pre> not <pre>one 3, two 2, three 1</pre> as before.

PUTS

RPL word PUTS can now have a format string, e.g.

```
"Hi there!" PUTS
prints
Hi there!
and
2 3 + "high %d" PUTS
prints
high 5
```

A COMPLETE GUI PROGRAMMING EXAMPLE

```
( GUI EXAMPLE
"ui.rpl" LOAD ( This contains definitions required by gui words
( Define a new data type 'pointer array'. The PARRAY takes one parameter which
( defines the number of items in the array to be created.
( MX gadget requires string labels in this format. For example: 10 PARRAY
( creates array of ten items.
: PARRAY
  <BUILDS 4 * ALLOT DOES>
;

( This stores a 32 bit address to a given slot of the array. For example
( 555 2 PSTORE stores the value 555 to the third slot.
: PSTORE
  4 * + STORE
;

( Global variables for holding addresses of gadgets to be created
VARIABLE aWindow
VARIABLE aButton
VARIABLE aCheckBox
VARIABLE aSlider
VARIABLE aMx
VARIABLE aString
VARIABLE aText
```

```
VARIABLE aText2
VARIABLE aText3
100 STRING sBuf

( A general usage word which prints out a given string using aText2 gadget
: PrintMsg
  UI_Done SWAP UITX_Text aText2 FETCH UI_SETATTRS
;

( Callbacks for gadgets
: cbButton
  ( local variables
  VARIABLE aTmp
  100 STRING sTmp

  ( fetch a string from the string gadget
  UI_Done aTmp UIST_String aString FETCH UI_GETATTRS

  ( and print it out
  aTmp FETCH "Loading %s" sTmp SPRINTF
  sTmp PrintMsg
;

: cbCheck
  IF
    UI_Done 1 UI_Disabled aSlider FETCH UI_SETATTRS
    UI_Done 1 UI_Disabled aMx FETCH UI_SETATTRS
    UI_Done 1 UI_Disabled aString FETCH UI_SETATTRS
    UI_Done 1 UI_Disabled aButton FETCH UI_SETATTRS
  ELSE
    UI_Done 0 UI_Disabled aSlider FETCH UI_SETATTRS
    UI_Done 0 UI_Disabled aMx FETCH UI_SETATTRS
    UI_Done 0 UI_Disabled aString FETCH UI_SETATTRS
    UI_Done 0 UI_Disabled aButton FETCH UI_SETATTRS
  ENDIF
;

: cbSlider
  "Slider %d" sBuf SPRINTF
  sBuf PrintMsg
;
```



```

: cbMx
  "Mutual Exclude %d" sBuf SPRINTF
  sBuf PrintMsg
;

: cbString
  PrintMsg
;

( Callback for window

: cbWindow
  PARAM ( parameters )
    VARIABLE iEvent
    VARIABLE iMouseX
    VARIABLE iMouseY
  ENDPARAM
  UIWM_Move iEvent FETCH = IF
    iMouseY FETCH iMouseX FETCH "Mouse moved to %ld %ld" sBuf SPRINTF
    UI_Done sBuf UITX_Text aText3 FETCH UI_SETATTRS
  ENDIF

  UIWM_LMBDown iEvent FETCH = IF
    "Mouse button clicked" sBuf CPY
    UI_Done sBuf UITX_Text aText3 FETCH UI_SETATTRS
  ENDIF

  UIWM_LMBUp iEvent FETCH = IF
    "Mouse button released" sBuf CPY
    UI_Done sBuf UITX_Text aText3 FETCH UI_SETATTRS
  ENDIF
;

( create a window
UI_Done
& cbWindow 0 0 400 200 "RPL Example Window" UI_WINDOW aWindow STORE

( create a read-only text
UI_Done
"RPL gadgets" UITX_Text
aWindow FETCH 0 150 15 100 12 "Header Text:" UI_TEXT aText STORE

( create a check box
UI_Done
0 UI_CB_Checked
aWindow FETCH & cbCheck 100 30 50 12 "Check Box" UI_CHECKBOX aCheckBox STORE

```

```

( create a string gadget
UI_Done
"spline" UIST_String
aWindow FETCH & cbString 100 45 150 12 "String" UI_STRING aString STORE

( create a slider gadget
UI_Done
0 UI_SL_Min
100 UI_SL_Max
20 UI_SL_Level
aWindow FETCH & cbSlider 100 60 50 12 "Slider" UI_SLIDER aSlider STORE

( create a mutual exclude gadget
5 PARRAY aLabels

"Choice 1" aLabels 0 PSTORE
"Choice 2" aLabels 1 PSTORE
"Choice 3" aLabels 2 PSTORE
"Choice 4" aLabels 3 PSTORE
0 aLabels 4 PSTORE

UI_Done
2 UIMX_Active
aLabels UIMX_Labels
aWindow FETCH & cbMx 100 75 150 12 "Mx" UI_MX aMx STORE

( create a button gadget
UI_Done
aWindow FETCH & cbButton 100 90 150 12 "Button" UI_BUTTON aButton STORE

( This text gadget shows gadget events
UI_Done
"Welcome to RPL" UITX_Text
1 UITX_Border
aWindow FETCH 0 100 105 150 12 "Gadget:" UI_TEXT aText2 STORE

( window events are printed here
UI_Done
"This is cool" UITX_Text
1 UITX_Border
aWindow FETCH 0 100 130 250 12 "Info:" UI_TEXT aText3 STORE

( realize gadgets
aWindow FETCH UI_REALIZE

( END OF RPL EXAMPLE )

```


5. MISCELLANEOUS

COMPATIBILITY

The binary format of many data types of Real 3D has been changed, but the program can convert modified data items automatically. Version 3 can read all v2.x files. Version 2.x can not always read v3.x files.

The major changes are:

- MORPHING_CLOSED method does not exist any more.
- The morphing method specific tags have been changed. Two new tags can now be associated with the morphing method:
 - IFLG (open/closed, selects also data to be morphed)
 - IMIT (Morphing Interpolation Type)
- Curves can now contain a knot sequence
- Fidelity added to bone description for C_SKELETON word.
- Freeform objects may contain skeleton fitting arrays.

The menu modifications mean that old RPL startup files are not fully compatible. If you have customized the RPL startup, take the modifications from the old file and add them to the file supplied with the new version. After that, verify that MENU word parameters match the new menu system.

The rendering engine algorithms have been modified, mainly for speed optimization reasons. This affects the image output. Therefore, do not render consecutive animation frames using different program versions, because even tiny changes may spoil the animation.

COMPARISON OF AMIGA / WINDOWS SPECIFIC FEATURES

This appendix and the actual Real 3D v.2.4 manual describe the version for Windows 3.1 and Windows NT platforms. The version for Amiga has almost identical functions. Nevertheless, there are some operating system specific features and differences, which are described in this chapter. See also the readme file of the Amiga version and the AmigaGuide help files for more information.

Installation

To install the Amiga version to hard disk, insert the disk 'Real3D_V3_Disk1' to a floppy drive, double click the 'Install' icon and follow the instructions. The full installation requires about 8 megabytes of hard disk space.

The installation procedure uses the standard Installer, (c) Copyright 1991-93 Commodore-Amiga, Inc. All Rights Reserved. Reproduced and distributed under license from Commodore.

Naming conventions

The File menu of the Windows version is called Project menu in Amiga.

The 'dialog' word is frequently used in the Windows environment instead of the word 'requester' which is more familiar to Amiga users.

Amiga's gadgets are often called as buttons, list boxes, edit boxes etc. in the Windows version.

Menu function differences

Project/Project/Insert Sections, Save Sections, Replace Sections

Amiga projects may contain multiple screens. Therefore, the sections requester contains an additional checkbox to control screen data IO operations.

Project/Materials/Window

In the Amiga version, some of the material window features can be accessed through menus, whereas the Windows version uses gadgets.

File/Materials/Material Select Window

The material select window which allows material selection from preview images is not available in Amiga.

Project/Windows/View superbitmap

Project/Windows/View borderless

Project/Windows/View dbuffered

These are Amiga specific view window types.

View_Superbitmap:

Open superbitmap View. This means the window can be moved over a much larger 'virtual' View using the scroll-bars at the bottom and right side. Objects can be created and modified, even rendered on any part of the superbitmap.

View_Borderless:

Open View covering the full screen. This has no visible borders and obscures the menu-strip. This window type is usually needed for rendering final images or animations where window borders are not desirable.

View_DBuffered:

Opens a double-buffered borderless View on a new screen. This enables wire-frame motion to be observed without seeing the intermediate drawing steps, so the image does not flicker. The new screen is always 4 color, 640 wide, maximum height and interlaced. This screen is a special private screen and no other windows can be opened on it. By default the View projection type is perspective.

Project/Windows/Animation

The automatic preview feature of the Windows version is not available.

In the windows version, you can select a view window to be saved using the 'Save' combo box. In the Amiga version, you can specify a screen to be saved with the following gadgets:

Screens & Saved

The list selector enables the one of the open Real 3D screens to be selected for saving to an IFF file after each animation frame. The name can also be typed directly into the Saved text gadget. If an external screen has been opened, then it can be selected and will be saved using the file settings selected with Project/External_Screen/Settings.

Save

If set, then the selected screen is saved to a file and Frame is incremented, after rendering of the current frame is completed.

Screen File

The name and path for the screen image files saved by the animation system.

In the Amiga version, it is possible to open the animation window to a separate screen. This is controlled by Settings/General/Anim. screen option.

Project/Windows/Palette

Opens a palette window. This is opened on its own private HAM screen. The Palette is used to control the R,G,B values of the current color and the 16 register color values. The current color is used whenever Real 3D requires a color value for a function e.g. Modify/Properties/Color.

The register colors give the user fast access to 16 different color definitions. Selecting one of the register color buttons makes the current color the same as the current value of the register color. The value of the register color is then changed using either the sliders or selecting from the color spectrum with the left mouse button. Holding the left button down while moving over the color spectrum changes the current color and the slider values continuously. There is no need to confirm these changes. The effects take place when the 'OK' gadget is pressed. It is also necessary to select the

settings required BEFORE using a function which uses the current color. If a Palette window is not currently open, then the last setting for the current color will be used.

Project/Windows/Screen

In the Amiga version, it is possible to use several screens at the same time. The screen window can be used to manage screens.

This opens a small window which enables control of the public screens used by Real 3D. It is opened automatically when a new screen is created to provide a 'handle' to the Real 3D menus, until other windows have been opened.

Gadgets of the screen window:

Default

Make active screen the default public screen.

Jump

Jump to next public screen. The control window will follow.

Close

Closes all Real 3D windows on the current screen then attempts to close the screen. If other windows prevent the closure, then a warning requester will be opened

Hijack

When enabled, windows and requesters opened for public screens will be hijacked and opened on the default public screen. This allows other applications which use public screens to be integrated into the Real 3D system. One possibility for this is to integrate a text editor for creating and editing the text for RPL.

Pop-to-front

When windows and requesters are opened on the default public screen, it will be brought to the front.

Project/Windows/Color Wheel

The standard color wheel window is also available in systems using Operating System v.39 or later. You can use it instead of the HAM palette window.

Project/Environment/Open screen

Open a new Real 3D screen using screen settings requester.

Project/Environment/Make Def. Pub.

Make currently selected screen the default public screen.

Project/Environment/Close screen

Closes the selected Real 3D screen.

Project/Environment/Close current

Closes any windows currently open on active screen and then attempts to close the screen. If other programs have opened windows on the Real 3D screen, then the screen cannot be closed and the user will be warned.

Project/Environment/Screen Palette

Opens a requester enabling the active screen colors to be changed. The screen register color to modify is selected from the color spectrum, then it can be adjusted using the RGB sliders. Two pre-defined palette definitions can be loaded by selecting either of the two button-gadgets. The 'COLOR SCALE' definition is used with Render Settings/Color Shading. Screen palettes are saved as part of the screens data section.

The screen palette requester contains two numerical gadgets. The purpose of the gadgets is to specify a range of palette entries to be used in greyscale shading. This way, the palette can be divided into two groups of colors: user interface colors and greyscale shading colors. The user interface colors can be freely modified without interfering the screen output quality.

The color range defined by the two gadgets is used when the GREYSCALE button is hit. Instead of spreading the greyscale gradient over the whole palette, only the colors within the range are modified. Also the greyscale rendering algorithms use colors in the range.

For example, you may set the 'Greyscale from' value to 4 and the 'To' value to 15 for a 4 bit plane (16 colors) screen. A press of the GREYSCALE button spreads the greyscale over the new range. After that, the first four colors can be modified to obtain a suitable user interface color combination.

Note: this feature is available only on screens opened by Real 3D. Note also that colorscale shading ignores the range definition, using all colors in the palette.

Environment/External screen

These functions enable a library for controlling an External screen, like a frame buffer, to be used. The sub menu items are:

Open

Open an external screen with its specified external screen library.

Close

Close external screen and its library.

Set Modes

This function allows settings of an external screen to be modified. Not all libraries need this function, and so will not produce a requester.

Settings

Opens external screen requester allowing the user to select external screen library and file format.

Gadgets of the External screen requester:

Library	Name of External screen library.
IFF24	IFF 24 file format used.
TARGA	Targa file format used.
TARGA+A	Targa file format with alpha information used for saving.
CUSTOM	External screen's own format.

Save

Save image from external screen using format specified by settings.

Create/Build Freeform/Font curve extractor

The TrueType font extractor is not available in the Amiga version.

Modify/Properties/Color

The color of the selected objects becomes the RGB values of the current color. If the Shift key is held down when this function is activated, the modal color requester is opened and the color of the first selected object is shown.

Modify/Color

This menu and its submenus are available in the Windows version only. In the Amiga version, use Modify/Properties/Color to change the color of the selected objects.

View/Grid/Define

The grid requester of the Amiga version allows the user to define a 16 bit 'Pattern' value, which defines a bit mask for grid lines. This makes it possible to use dotted lines, etc.

View/Save Window

Not available in the Amiga version. Only complete screens can be saved using Project/Environment/Save screen function.

Settings/View Window color

This menu does not exist in the Amiga version. Use the Project/Environment/Screen palette function to set the background color (the first palette entry) to a desired value.

Tool/Icons

The Auto size option of the Windows version is not available.

INDEX

SYMBOLS

[&] 81

3DStudio files

exporting to 3DS 47

loading 33

A

Additive option (glow) 50

ADDMENU 68

Alpha channel output in IFF file rendering 49

Angle

constraint of skeleton joint 40

of helix 40

of rotation 37

Angle scope handler 17, 34

Animate/ 52

Animation 18-32

creating an animation, example 7

rendering 10

restoring after playback 32

Animation system 55-66

Animation window (Amiga/Windows) 89

Anti-aliasing 34, 50. *See also* Super sampling

Attributes

geometric 44

Modify/attributes dialog 43

Settings/Attributes 54

skeleton attributes 40

Auto Box Rendering 48

Auto Track 55

B

Backdrop image blitting 51

Blt Backdrop 51

Blurring sensitivity 51

Borderless view 89

Bounding Box 43

Brightness

of glow 34, 50

Bumps

density and height 34

C

Camera View 48

Canceling

creation/modification operation 37

Center (view tool window) 37

Clockwise normals option (freeform type dialog) 46

Closed morphing 60

COG 60. *See also* Pivot (key editor)

Col.repl 35

Color

dither color handler 35

material color 35

Modify/Color (Windows version) 92

Modify/Properties/Color (Amiga version) 92

morphing 61

window color 92

COMPARE 68

Comparison of Amiga / Windows specific features 88

Compatibility 87

Controls/ 40-42

Convert to Triset 47

Create (key editor) 56

Create Key (key editor) 56

Create/ 38-43

Creating

a primitive 36

CS_DEFINE 69

CS_ROTATE 69

CS_SCALE 69

CS_TRANSFABS 69

CS_TRANSLATE 70

Cumulative shrink wrapping 65

D

DATETIME 70

DBuffered view 89

Delete Key (key editor) 56

Depth
 of a primitive 37
 Depth scope handler 34
 'di' Rpl variable 35
 Discrete interpolation 60
 Distance anti-aliasing 50
 Distance blur 51
 Distribute (key editor) 56
 Dither color handler 35
 Drawing Settings 51
 DXF format
 exporting in 51

E

Edge X/Y 33
 Edit (key editor) 57
 Editing animation 53
 Envelope window 21
 Envelopes 57
 key editor 56
 Export DXF 51
 External screen (Amiga version) 91
 Extras/ 53-54

F

Fade 46
 animating 23
 FCLOSE 70
 FEOF 70
 FGETS 70
 Fidelity 41, 47
 an example 26
 File/ 33-38
 Fixed
 skeleton type 42
 Fixed size option (glow) 50
 FOPEN 71
 FPUTS 72
 Fractals 15
 Fractals/ 43
 Frames 55
 FREAD 72
 Free
 skeleton type 40
 Freeform 46-47
 subdividing into groups 53

Freeform modelling
 example 12-15
 Freeform/ 51
 Freq X and Y 33
 Friction
 of skeleton joint 41
 FSEEK 72
 FSKF tag 41
 FTELL 72
 FWRITE 73

G

General settings 54
 Geometry
 geometry dialog 44
 morphing 61
 Glow 34, 50
 an example 16
 Gradient. *See* Edge X/Y
 Gravity (surface method) 64
 Grid requester (Amiga) 92
 Group 38
 deselecting 53

H

Helix 40
 Hierarchical skeleton 42, 63
 an example 27
 Hierarchical animations
 an example 20

I

IFAD tag 23, 46, 62
 IFF file rendering 49
 IFLG tag 59, 60, 65
 IIND tag 63
 IINH tag 39
 Image formats 36
 IMIT tag 59, 60
 Inherit option 19, 39
 Installation (Amiga/Windows) 88
 Interlace fields 48
 Interpolation 56, 60. *See also* IFLG tag; IMIT tag
 Inverse kinematics 52, 63
 Invisible checkbox (key editor) 55

ISKE tag 64
 ITRA tag 65

J

JPEG images 36
 JPEG support 49

K

Key editor 18, 55
 Keyframe animations 52, 59
 an example 9, 20
 Knot sequence 55. *See also* Distribute (key editor); Path
 animations

L

Landscape 43
 LEN 73
 Linear/B-Spline/Discrete (key editor) 56
 Link 38
 Links (surface method) 64
 Loading
 landscapes/ trees 43
 render settings 49

M

Mapping 35
 defining 8
 Material 16-18
 color 35
 defining 8
 morphing 61
 Materials/Purge 36
 Materials/Window 33-36
 Menu function differences (Amiga/Windows) 88
 Menu functions 33-54
 Method
 Create/Method menu 39
 key editor 56
 Modelling 10-16
 freeform, an example 12
 Modify/ 43-47
 Modifying
 a primitive 36
 Morphing 60
 an example 23
 skeletons, an example 29

N

Naming conventions (Amiga/Windows) 88
 NCAT 73
 NCPY 73
 Next (key editor) 56
 "No RPL busy requester" checkbox 54
 NOISE 74
 Non-cumulative shrink wrapping 65

O

O_GETSEL 69
 O_MAKENAME 74
 Objects/Load 33
 Outl. Depth 37

P

Palette window (Amiga version) 89
 Parallel Shrinking 13
 Path animations 59
 an example 18
 Periodic (key editor) 56
 Pivot (key editor) 57
 Pivot point 60
 an example 22
 Position coordinates 37
 Post effects 17, 49
 PPM images 36
 Prev (key editor) 56
 Project/Insert Sections, Save Sections, Replace Se 33
 Properties/ 43-46
 PUTS 82

R

Reflect 46
 Relative texture coordinates 35
 REMOVE 75
 Render/Settings 48-51
 Rendering
 animation 10
 Auto Box 48
 IFF file 49
 quality 34. *See also* Super sampling
 time. *See* Auto Box Rendering; Super sampling
 Repeat Last 54

- 'ro' RPL variable 34
- Rotate (envelope editor) 58
- Roughness bump handler 34
- Round 47
 - example 10

S

- Sampling density 48
- Saving
 - landscapes/ trees 43
 - render settings 49
- Scale (envelope editor) 58
- SCANDIR 76
- Scope handler 34
- Screen Palette 91
- Screen window (Amiga version) 90
- Settings/ 54
- Shear (envelope editor) 58
- Show selected objects 54
- Shrink wrapping 53, 64
 - an example 32
- SIDE tag 38, 63
- Simple skeleton method 63
- Skeleton 40–43, 52
 - an example 24
 - method 63
 - morphing, an example 29
- SMAT tags 36
- Special/Reflect 46
- SPRINTF 81
- STOP (key editor) 56
- Structure/ 38–39
- Sub skeletons 42
- Super sampling 48
- Superbitmap view 89
- Surface coordinates 11
- Surface method 64
 - an example 30
- Surface to Groups 47

T

- Tags
 - morphing 62

- Texture
 - antialiasing 34
 - frequency 33
 - gradient 33
 - one-sided 34

- Time
 - inheritance 39
- Transform method 66
- Translate (envelope editor) 58
- Tree 15, 43
- Triset
 - converting objects to trisets 47
 - shading 33
- Type
 - of freeform 46
- Type/Surface 47

U

- UI_BUTTON 77
- UI_CHECKBOX 77
- UI_DELETE 80
- UI_GETATTRS 80
- UI_MX 79
- UI_REALIZE 81
- UI_SETATTRS 80
- UI_SLIDER 79
- UI_STRING 78
- UI_TEXT 78
- UI_WINDOW 76

V

- Vectors 53
- View/ 47–51
- ViewTool window 36
- VMOV tag 59
- VROT tag 59
- VSHE tag 59

W

- Window color 92
- Windows/ViewTool 36–38



REAL 3D V.3 Some highlights

Interface

- Several functions have been introduced to allow users easier access to commonly used functions.
- The view tool window provides quick access to object creation and modification tools.
- Object geometry can now be edited directly at any time. Points, depth, etc. can now be edited numerically through this new window.

Fast rendering

- The rendering engine has been improved and a new 'auto box' rendering mode introduced. On complex scenes this can reduce render time by half.

Pick at surface

- This is a new type of view mode, user can now draw curves on the surface of other objects. Using the new mode objects can be joined together with spline patches seamlessly or shapes extruded from a surface.

Background Blitting

- All view windows can have a full colour backdrop image in wireframe mode for rotoscoping, perspective matching, etc.

Post Processing Effects

- A new open interface for post processing has been included. Effects such as lens flares, glows, etc. can now be added to any scene. One of the included effects, Material Glow, allows the user to create all types of effects such as fire, laser beams, etc.

Animation

- The animation system has been extended, new methods include keyframing, drag, and shrink wrapping.
- The drag method allows easy creation of animation where objects need to 'walk' across a surface. The user defines impact points on the object and REAL will 'walk' the objects across a given surface by those points.
- Animated shrink wrapping provides access to effects such as footprints in the snow. Any animated object can be moved across a mesh leaving an impression as it goes.
- Skeletons can now be nested allowing them to be branched. Constraints have been introduced so each join of a skeleton can be fully controlled by the user.
- Object fades. A new attribute 'fade' has been introduced allowing objects to have a transparency level which is independent to any materials. The fade level can be morphed allowing all kinds of effects to be created simply and quickly.



REALSOFT